UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Herramienta visual colaborativa para la mejora de la usabilidad

AUTORES: Verónica Soledad Murga

DIRECTOR: Alejandra Garrido

CODIRECTOR: Sergio Firmenich

ASESOR PROFESIONAL: ----

CARRERA: Licenciatura en Sistemas

Resumen

En la actualidad más empresas se insertan en el mercado virtual y ya es bastante común que los usuarios busquen lo que necesitan en la web. Es por este motivo que ha incrementado la necesidad de enfocarse en aspectos de usabilidad y accesibilidad que antes no eran tenidos en cuenta. Creemos que es muy importante que el usuario pueda acceder a la información de manera sencilla e intuitiva. Si bien hoy en día las empresas comenzaron a enfocarse bastante en las cuestiones de usabilidad, no alcanza para cubrir todos los aspectos necesarios para satisfacer a todos los tipos de usuarios que pueden llegar a utilizar la aplicación. En este trabajo proponemos herramientas para ayudar a mejorar la usabilidad de las aplicaciones web y que cada usuario pueda adaptarlo a su necesidad sin tener que esperar que el dueño del sitio lo haga. Nos enfocamos en desarrollar herramientas intuitivas para ser utilizadas por voluntarios novatos en el área de la informática permitiendo que cualquier usuario que quiera colaborar y enriquecer la comunidad sea bienvenido. Las herramientas desarrolladas permitirán reportar problemas de usabilidad en carácter de usability smells y a la vez solucionar dichos reportes. Nos enfocamos en el modelo de crowdsourcing permitiendo a cualquier usuario participar y enriquecer la comunidad.

Palabras claves

Usabilidad, accesibilidad, crowdsourcing, usability smells, refactorings, usability refactorings, webextension, Javascript, aplicaciones web, Python, API, Client side web refactoring, modelo de base de datos.

Trabajos realizados

-Herramienta colaborativa para reportar usability smells.
-Herramienta colaborativa para la gestión de reportes de usabilidad.
-Herramienta colaborativa para la resolución de los reportes realizados.
-Desarrollo del refactoring Split Page y Distribute Menu.
-Desarrollo de API para comunicarse con Kobold.

Conclusiones

El trabajo desarrollado permite mejorar la usabilidad de las aplicaciones webs con la ayuda de la comunidad. De esta manera mejorar la experiencia de los usuarios en los sitios web adaptándolos a sus necesidades. Es muy importante destacar la importancia de la ayuda de la comunidad para poder cumplir el objetivo realizando los reportes y solucionandolos.

Trabajos futuros

-Perfiles para usuarios en Usability Smell Manager.
-Clasificación de refactorings con orden de ejecución.
-Permitir al voluntario desarrollar refactorings propios.
-Mejora en el guardado y presentación de resolución de reportes.
-Internacionalización de las aplicaciones.

Fecha de la presentación: Noviembre 2019

Agradecimientos

Este trabajo no hubiera sido posible sin la inmejorable ayuda y dedicación de mi directora, Alejandra. Gracias por las invaluables enseñanzas y aportes que me diste. Tampoco habría sido posible sin la ayuda y el apoyo de Sergio. Gracias al equipo del LIFIA, en especial a Julián y Juan Cruz por la ayuda y el soporte.

El camino fue largo y difícil, nada de esto hubiera sido posible sin el incondicional apoyo de mi familia. Mi mamá, mi papá y mis hermanos, a ellos, gracias infinitas.

Cómo no agradecer también a todas las hermosas personas que conocí a lo largo de este camino. A todos mis compañeros y amigos de la facultad, en especial a Federico, Juan Pablo, Luciano, Mariángeles, Marcia,Mariel. Gracias por ayudar a que este largo trayecto sea el más maravilloso de todos!

Y sobre todo a mi hermoso compañero de vida, Esteban. Sin vos esto no habría sido posible.

CAPÍTULO 1

Introducción

En la actualidad, la tecnología se ha convertido en un pilar muy importante especialmente para el crecimiento y desarrollo de las empresas. Cada vez más empresas la utilizan para incrementar sus ingresos. Las aplicaciones web son las más elegidas y las más rentables. A pesar de esto, durante su desarrollo, no se enfocan en aspectos de usabilidad, es decir, en cuán amigable es el sitio para los usuarios. En consecuencia, provoca que la utilización de estos sea confuso y dificultoso para los usuarios, lo que podría generar pérdida para la compañía.

Según Steve Krug en su libro “Don’t make me think” [Krug05], cuando habla sobre el desarrollo de un sitio web usable, expresa que el usuario debería ser capaz de entenderlo, es decir saber qué es y cómo usarlo, sin necesidad de esforzarse para pensar sobre esto. El sitio es agradable al usuario si éste se explica por sí solo o si bien las páginas son evidentes. Todo lo que haga al usuario pensar de más provocará que este probablemente no quiera utilizar más el sitio y vaya en busca de otros que sean más fáciles de utilizar para encontrar lo que necesitan.

El rol del usuario ha tomado gran importancia en los últimos años para los sitios web ya que éste no sólo utiliza el sitio para informarse sino también que puede utilizarlo para tomar beneficios del mismo. Ejemplo de estos casos son sitios e-commerce, redes sociales, home banking, entre otros. Gracias al incremento de la interacción del usuario en dichos sitios web ha surgido la necesidad de enfocarse en conceptos relacionados con la usabilidad que antes no se tenían en cuenta [Brhel15]. A pesar de esto la usabilidad es ignorada por la mayoría de los desarrolladores web pudiendo provocar que el sitio sea poco entendible para el usuario y en consecuencia éste último pierda interés en interactuar con el sistema.

Muchos problemas se solucionarían si se tuviera en cuenta desde el inicio la usabilidad y la accesibilidad. Pero como es sabido esto no es así y sería provechoso permitir que la misma comunidad de usuarios pueda proponer soluciones y compartirlas.

Esto lleva a la necesidad de crear una comunidad para la resolución de problemas de este tipo. Esto consiste en exponer el problema y un grupo de personas ofrecen una solución y la mejor de todas es la elegida. Esta técnica se denomina *crowdsourcing*.

El crowdsourcing [Brabham14] es un recurso en línea que sirve para la resolución de problemas a través de una comunidad. Esta última aporta sus conocimientos y experiencias para encontrar la mejor solución al problema.

1.1 Objetivo

La usabilidad y accesibilidad son características muy importantes de las aplicaciones web que en general no son tenidas en cuenta por el alto costo que implica su evaluación y reparación, sobre todo para las pequeñas y medianas empresas. Desde hace unos años se

ha propuesto la técnica de refactoring para mejorar la usabilidad y la accesibilidad [Garrido11], y se desarrolló un *framework* para crear estos refactorings en Javascript llamados "Client Side Web Refactorings" (CSWR), porque los mismos se instalan del lado del cliente [Garrido13a]. Asociado a los refactorings de usabilidad se definió el concepto de *usability smell* como una indicación de un problema de usabilidad que puede solucionarse a través de *refactorings* [Garrido11]. De manera similar se definió el concepto de *accessibility smell* [Garrido13b]. El problema que aún tiene esta tecnología es que queda reservada para unos pocos que la conocen y puedan extenderla programando nuevos refactorings en Javascript.

En este contexto esta tesina tiene como objetivo general proponer un método y herramientas de fácil adopción que incrementen la comunidad de voluntarios que desarrollen refactorings de forma colaborativa, y así mejorar la usabilidad de los sitios web masivamente.

Para lograr esto se atenderán los siguientes objetivos específicos:

- Crear una plataforma que reciba *bad smells* (BD) de usabilidad o accesibilidad en un formato predeterminado por parte de los usuarios, e instrumente un mecanismo que permita a cualquier otro usuario voluntario seleccionar un BS para darle solución.
- Desarrollar una herramienta que pueda sugerirle al voluntario posibles soluciones a un BS dado en términos de posibles refactorings y permita que un voluntario sin conocimiento de programación pueda ejecutar refactorings utilizando programación visual.
- Permitir al usuario con conocimientos en Javascript desarrollar refactorings y proponerlos como solución a un determinado BS.

1.2 Motivación

La usabilidad en los sitios web hace referencia a qué tan fácil es para el usuario llegar a completar su objetivo con la aplicación [Nielsen99]. La usabilidad es un punto clave dentro del campo del comercio electrónico, ya que si el usuario no puede encontrar lo que busca, no consumirá el producto o, peor aún, dejará de utilizar el sitio web en busca de otro que le ofrezca lo mismo y pueda encontrar lo que busca fácilmente [Nielsen06].

Existen en la actualidad algunas propuestas para involucrar a los usuarios, sobre todo en ayudar a mejorar la accesibilidad de un sitio agregando información semántica [Takagi08], y plataformas para crear *scripts* del lado del cliente, como Grease Monkey. Existen otras herramientas para hacer *tests* de usuario para encontrar problemas de usabilidad [Carta11], pero los reportes que genera deberían ser leídos por expertos en el área y las modificaciones sugeridas realizadas por los desarrolladores del sitio web. Esto provoca que sólo los que poseen conocimiento técnico puedan favorecerse con la herramienta. Tampoco existen herramientas que le permita al voluntario, como usuario de la aplicación web, reportar malos olores para mejorar tanto la usabilidad como la accesibilidad del sitio y sugerir una solución para los mismos, de tal forma que el resto de la comunidad

pueda hacer uso de estos. Esta técnica se conoce como crowdsourcing [Howe08], donde participan los mismos usuarios en solucionar los problemas y en este caso se orientaría a mejorar la usabilidad.

Actualmente existen muchos enfoques para proponer y aventurar a los usuarios a participar en el proceso de evaluación de usabilidad de un sitio web. Es muy importante la opinión de estos ya que son los que van a utilizar el sitio en su mayoría de tiempo. Algunas son para usuarios inexpertos [Takagi08] y otras donde el usuario debe tener mayor conocimiento de programación [Grigera17].

Se utilizará la técnica de refactoring para aplicar las modificaciones propuestas por el voluntario. Esta técnica consiste en realizar mejoras en la página web sin modificar su funcionalidad [Fowler99]. Este trabajo se enfocará en aplicar refactorings que solucionen problemas tanto de accesibilidad como de usabilidad. Se tomará como base los fundamentos utilizados en el *framework* CSWR [Garrido13a], el cual permite realizar adaptaciones en la página web aplicando *refactorings* escritos en Javascript del lado del cliente sin necesidad de ser el dueño del sitio. Dicho *framework* aún se encuentra en desarrollo. Una de las desventajas de esta herramienta es que solo aquellos voluntarios con conocimiento del lenguaje Javascript pueden desarrollar los *refactorings*, lo cual restringe la cantidad de voluntarios a gran medida. El propósito de este trabajo es quitar esta restricción, recurriendo a la programación visual [Ko11] para que cualquier persona pueda participar en la mejora de la usabilidad web.

1.3 Contribución

Con el objetivo de permitir que la misma comunidad de usuarios pueda reportar problemas de usabilidad en las aplicaciones web que utiliza y corregirlos fácilmente, de forma colaborativa y masiva, este trabajo se basa en los conceptos de *bad smells* (BS) y *refactoring* aplicados a la usabilidad y accesibilidad.

En particular, hemos tenido en cuenta los conceptos abordados en un *framework* preliminar existente de CSWRs [Garrido13a] que permite aplicar *refactorings* del lado del cliente en aplicaciones web existentes para desarrollar una herramienta que mejora la usabilidad en las aplicaciones web. Gracias a esto, la comunidad de usuarios puede refactorizar un sitio sin necesidad de esperar a que el dueño del mismo lo haga.

Si bien dicho *framework* es robusto y actualmente se desarrollaron varios *refactorings* genéricos, los cuales se pueden aplicar a cualquier sitio, y varios *refactorings* instanciados, los cuales se aplican a un dominio en específico, aún quedan muchos por desarrollar y, a su vez, no existe un soporte para dichos *refactorings*, es decir, no existe un lugar donde alojarlos y gestionarlos. La herramienta se instala como extensión del navegador, pero actualmente sólo es soportado por Firefox versiones inferiores a la 57.

Además se han utilizado catálogos de BS y CSWRs existentes, que a su vez se extendieron y enriquecieron en el contexto de este trabajo para facilitar su utilización por usuarios web que no sean expertos en usabilidad. A partir de estos conceptos se han construido herramientas que permiten:

- Que cualquier usuario pueda reportar un problema de usabilidad o accesibilidad en forma de BS sobre cualquier sitio web, y seleccione de forma visual todos los elementos de la página que hace al reporte del problema.
- Que cualquier otro usuario con cierto conocimiento de las soluciones de usabilidad y accesibilidad apropiadas pueda proponer soluciones a los BS reportados a través de la programación visual de CSWRs.

También se diseñó un repositorio para alojar los refactorings y los reportes realizados y de esta forma poder gestionarlos. Por último se desarrolló una API que permite recibir malos olores de usabilidad y los refactorings necesarios, mediante un estándar predefinido, desde la aplicación desarrollada para este fin [Grigera17].

Uno de los puntos más importantes a destacar de la nueva herramienta desarrollada es que el usuario podrá ser partícipe en todo momento tanto para reportar *usability smells* como para proponer soluciones a los reportes sin la necesidad de ser un experto en el área mediante una interfaz visual muy amigable. Además los reportes pueden realizarse en tiempo real, es decir, no es necesario abrir una aplicación externa al navegador web. El usuario puede navegar libremente internet y, cuando lo considere necesario, abrir la extensión del mismo navegador y realizar un nuevo reporte.

1.4 Organización de la tesina

Capítulo 1: en este capítulo se presentan los objetivos, motivación, contribución realizada del trabajo y la organización de la tesina.

Capítulo 2: en este capítulo se presenta el marco teórico necesario para comprender el desarrollo de este trabajo junto con la introducción al framework CSWR desarrollado para la aplicación de refactorings del lado del cliente del cual se basa esta tesina. También se realiza una introducción a la aplicación Kobold con la cual se trabajará en conjunto.

Capítulo 3: en este capítulo se describe la arquitectura de las aplicaciones y cómo funciona cada una de ellas para cumplir con el objetivo propuesto, mejorar la usabilidad de las aplicaciones web. También se presenta el modelo de base de datos que usarán las tres aplicaciones para cumplir con la finalidad de reportar y solucionar un determinado *usability smell.*

Capítulo 4: en este capítulo se describe la primer herramienta involucrada en el proceso, *Usability smell reporter*. Esta es la encargada de permitirle al usuario realizar los reportes de *usability smells* y de ejecutar de manera local los reportes ya solucionados. Se explicará en detalle su funcionalidad técnica y su funcionalidad como usuario final.

Capítulo 5: este capítulo presenta la herramienta intermedia entre ambas extensiones, *Usability smell manager*. Esta nos permite gestionar el listado de reportes y a su vez gestionar la base de datos central de todo el proceso. Se explicará en detalle su funcionalidad técnica y su funcionalidad como usuario final.

Capítulo 6: este capítulo describe la última herramienta del circuito, *Usability smell fixer*. Es la que permitirá al usuario, luego de haber seleccionado un reporte, proponer una solución. Se explicará en detalle su funcionalidad técnica y su funcionalidad como usuario final. Finalmente se presentarán y explicarán los dos refactorings desarrollados en este trabajo como contribución a los ya existentes en el catálogo: Split Page y Distribute Menu.

Capítulo 7: aquí se presentan dos casos de estudios de la herramienta. Cada caso es acompañado con imágenes y descripciones paso a paso de cómo realizar el circuito completo de reporte y resolución de un determinado *usability smell* en una página específica.

Capítulo 8: capítulo final donde se presentan las conclusiones alcanzadas por este trabajo así como propuestas de trabajos a futuro.

Anexo A: en este anexo se describe la API Usability Smell desarrollada para interactuar con los aplicaciones y la base de datos central durante todo el proceso de reporte y resolución de BS.

Anexo B: en este anexo se describe en detalle la API que Kobold provee para realizar la comunicación e intercambiar requerida para realizar los reportes y las resoluciones de los mismos.

CAPÍTULO 2

Trabajos relacionados y Marco teórico

2.1 Usabilidad

La usabilidad hace referencia a qué tan fácil es para el usuario utilizar un producto determinado para alcanzar el objetivo concreto [Nielsen99].

La ISO 9241-11 describe que la usabilidad y el diseño de la experiencia de usuario trata sobre los productos diseñados para que sean eficaces, eficientes y satisfactorios. Específicamente, la ISO define la usabilidad como la "medida en que un producto puede ser utilizado por usuarios específicos para lograr los objetivos especificados de manera eficaz, eficiente y satisfactoria en un contexto específico de uso" [ISO11].

El aspecto de usabilidad sobre el cual nos centraremos en este trabajo, será la usabilidad web. Es muy importante diseñar aplicaciones web que sean fáciles de utilizar por el usuario ya que "si el cliente no puede encontrar el producto, entonces él o ella no lo comprará" [Nielsen99].

Por otro lado Krug define la usabilidad en su libro [Krug05] como la capacidad de entendimiento de un usuario promedio sobre un sistema común, es decir, que cualquiera sea el nivel técnico de conocimiento del usuario, éste deberá ser capaz de utilizar el sistema sin ningún tipo de problema o complejidad que pueda provocar que no quiera volver a hacerlo.

El objetivo principal de las empresas que se preocupan por mejorar la usabilidad de sus productos es lograr la fidelización de los usuarios, para que éstos sigan utilizando el producto y de esta manera también lo recomienden. Es muy importante que la utilización del producto sea fácil, intuitiva y amigable para el usuario. Si el usuario pierde interés, seguramente abandone el producto y no quiera volver a utilizarlo lo que significa una pérdida importante para la compañía.

Existen cinco principios básicos importantes a tener en cuenta a la hora de evaluar la usabilidad sobre el producto en cuestión:

- **Facilidad de aprendizaje:** hace referencia a la facilidad que tendrá el usuario a la hora de realizar una tarea simple en el sistema. Los usuarios que son nuevos utilizando el sistema deberán ser capaces de realizar sin problemas estas interacciones.

- **Eficiencia:** hace referencia a la curva de aprendizaje que tiene el usuario sobre el sistema provocando que éste tenga una interacción más efectiva y de ese modo evaluar la eficiencia del producto.

- **Flexibilidad:** facilidad con la que el sistema y el usuario pueden intercambiar información. Hace referencia también a la capacidad de mantener al usuario

informado paso a paso de modo que no pierda el hilo de la ejecución y no olvide lo que estaba haciendo.

- **Errores:** el sistema deberá ser capaz de recuperarse ante cualquier error e informar al usuario lo ocurrido y que éste tenga la posibilidad de aprender de los errores y continuar con la operación para poder finalizarla.

- **Satisfacción:** es el resultado de conformidad que tuvo el usuario a la hora de interactuar con el producto. Se deberá evaluar el diseño y la facilidad de uso teniendo en cuenta los pasos realizados para realizar una acción.

Usabilidad en las aplicaciones web

Es muy importante que se tengan en cuenta los conceptos de usabilidad a la hora de desarrollar una aplicación web. Para considerar que algo funciona bien una persona experta o no debe tener la capacidad de utilizarlo sin ningún problema, sin siquiera pensar demasiado a la hora de realizar algún tipo de acción sobre el sistema [Krug05]. Krug expresa también una serie de consejos sobre cosas a tener en cuenta para que la usabilidad de la aplicación se explote al máximo. Algunos de ellos hacen referencia a que una aplicación debe ser autoexplicativa, es decir, explicarse por sí misma. Con solo verla el usuario deberá ser capaz de saber de qué trata la aplicación e intuitivamente navegarla. El usuario no debe tener que pensar demasiado ya que eso provocará frustración y el deseo de abandonar la aplicación. Las operaciones en la aplicación deben ser sencillas y rápidas. También hace énfasis en la importancia de que el contenido del sistema sea el justo y necesario. El sistema debe mantener actualizado al usuario sobre el estado de éste mediante notificaciones.

El usuario es lo más importante y hay que cuidarlo, por ello hay que ofrecerle un sitio amigable y accesible. Para esto han surgido diferentes métodos de evaluación de la usabilidad, que Fernández et al. clasifica como *métodos de inspección* y *métodos empíricos* [Fernandez11]. En el caso de los métodos de inspección, un experto utiliza heurísticas conocidas o utiliza recorridos cognitivos poniéndose en el lugar del potencial usuario y tratando de anticiparse a todos sus movimientos según distintos perfiles. Los métodos empíricos se basan en hacer pruebas de usabilidad con usuarios finales, y son los más utilizados.

Métodos de evaluación de la usabilidad

La usabilidad a lo largo del tiempo ha ido mejorando y se ha dado mayor importancia a este concepto debido a que en la actualidad casi todo el mercado se encuentra en internet y sus aplicaciones. Las empresas están realizando mayores inversiones en cuestiones de usabilidad pero aún no es suficiente, existen muchas falencias y no es posible acomodar las

aplicaciones teniendo en cuenta los aspectos necesarios para abarcar todos los perfiles de usuarios posibles.

En la actualidad existen diferentes métodos para evaluar la usabilidad de un sistema y así saber qué debe tenerse en cuenta para mejorarlo. Una de las herramientas que se pueden utilizar para medir la satisfacción de los usuarios es la escala de usabilidad (SUS - System Usability Scale) [Brooke96]. La escala ofrece diez preguntas importantes organizadas en un cuestionario que permiten obtener una visión subjetiva de los usuarios con respecto a la usabilidad de un sistema. Cada pregunta deberá tener un número finito de respuestas (en general 3 o 5) basadas en una escala Likert[1]. La desventaja de esta técnica es que es imprescindible la utilización de un grupo determinados de encuestados y la realización de un análisis exhaustivo de los resultados de la encuesta, que solo puede realizar un experto en usabilidad. Además no es fácil mapear los resultados de las encuestas a problemas concretos y soluciones específicas.

Uno de los conceptos más importantes a la hora de tener en cuenta para la prueba de usabilidad en el diseño de interfaces son las heurísticas de Nielsen [Nielsen90]. Nielsen las llama "heurísticas" debido a que se trata de conocimiento natural y general y no pautas estrictas y específicas de usabilidad. Se definen diez heurísticas en total. A continuación se presenta y explican cada una de estas.

1. **Visibilidad del estado del sistema**: el estado del sistema siempre tiene que estar visible al usuario. Se le debe mantener informado sobre cualquier tipo de cambio dentro de un tiempo razonable.

2. **Coincidencia entre el sistema y el mundo real**: se debe utilizar el lenguaje del usuario, es decir, tener en cuenta a quién va dirigido el sistema. Evitar tecnicismos para hacer que la información se vea lo más natural posible.

3. **Control y libertad del usuario**: muchas veces los usuarios cometen errores involuntarios y tratarán de deshacerlos. Es importante ofrecerles la posibilidad de hacerlo soportando las acciones de hacer o rehacer para mantener así el control del sistema.

4. **Consistencia y estándares**: tratar de seguir los estándares para que el usuario no deba pensar demasiado. Seguir las convenciones de la plataforma.

5. **Prevención de errores**: se debe tratar de prevenir los errores que puedan llegar a ocurrir. No se trata solo de utilizar mensajes lindos sino que el objetivo principal es prevenir errores. Utilizar de manera estratégica e inteligente cada mensaje. Solicitar confirmaciones antes de realizar cualquier tipo de acción que no puede ser deshecha o sea una acción sensible.

[1] Escala Likert: se utiliza para especificar el nivel de acuerdo o desacuerdo a un ítem o pregunta de una encuesta. Cada elemento de la encuesta puede evaluarse por separado por lo tanto cada conjunto de respuestas puede tener resultados distintos dependiendo el encuestado.

6. **Reconocimiento antes de recordar**: hace referencia a minimizar la carga de datos en la memoria del usuario haciendo objetivos, acciones y opciones visibles. El usuario no debería tener que recordar información cuando pasa de un lado a otro dentro de la aplicación. Las instrucciones de cómo utilizar el sistema deberían estar siempre a mano.

7. **Flexibilidad y eficiencia de uso**: ofrecer flexibilidad de uso tanto a los usuarios expertos como a los novatos. Para usuarios expertos permitirles utilizar atajos para realizar las acciones más rápidas y a los usuarios novatos ofrecer una interfaz sencilla pero con la posibilidad de aprender con la frecuencia de uso.

8. **Diseño estético y minimalista**: la aplicación no debe estar sobrecargada de contenido. No se debe incluir información que no es relevante para el usuario en los diálogos ya que esto podría provocar que el usuario se pierda o que la información importante pase a segundo plano.

9. **Ayudar a los usuarios para reconocer, diagnosticar y recuperarse de los errores**: los mensajes de error deben ser precisos y estar escritos en el lenguaje que el usuario utiliza y entiende. Es importante que se sugieran soluciones, si las hay, a los errores para permitirle al usuario recuperarse rápidamente.

10. **Ayuda y documentación**: aunque el sistema pueda utilizarse sin documentación, es importante tenerla y que el usuario tenga acceso a esta fácilmente. Debe ser concentra y precisa para que el usuario pueda entenderla. También una buena solución es agregar un apartado de preguntas frecuentes.

Accesibilidad

Otro concepto importante dentro de usabilidad es la “accesibilidad”. Esta hace referencia al uso de los sitios web por personas con capacidades reducidas. Según la W3C aunque la accesibilidad se centra en las personas con discapacidad, muchos requisitos de accesibilidad también mejoran la usabilidad para todos [W3C]. La accesibilidad puede ser incluída para ambos conceptos. Existen requisitos que son más específicos para las personas con discapacidad, por ejemplo, asegurar que los sitios web funcionen bien con tecnologías de asistencia, como lectores de pantalla que leen en voz alta páginas web, lupas de pantalla que agrandan las páginas web y software de reconocimiento de voz que se utiliza para introducir texto. Otros requisitos que también son principios de usabilidad generales, se incluyen en los requisitos de accesibilidad, ya que pueden ser barreras significativas para las personas con discapacidad [Sullivan00]. Por ejemplo, un sitio web que se desarrolla para que pueda ser utilizado sin un ratón es una buena usabilidad; Y el uso sin un ratón es un requisito de accesibilidad porque las personas con algunas discapacidades físicas y visuales no pueden usar un ratón en absoluto.

2.2. Refactoring

Los refactorings se definieron inicialmente como técnica para solucionar problemas en el código fuente del software, siempre y cuando no se modifique la funcionalidad original del programa [Fowler99]. Cada refactoring aplica a un dominio en particular, es decir, soluciona un problema específico en el código, al cual se define como *bad smell* (BS). La presencia de un *bad smell* es un indicador de la necesidad de aplicación de un refactoring. Existe un catálogo de *bad smells* [Fowller99] donde se describe el escenario posible de aparición y cuál es el refactoring a aplicar para su solución. Esto ayuda al programador a estar atento y ante cualquier situación como esta poder aplicar fácilmente la mejora a su código.

Un ejemplo de BS es “Método extenso”, el cual ocurre cuando un método es demasiado largo y se hace casi imposible entenderlo y por lo tanto será muy difícil de mantener. El refactoring propuesto para este *bad smell* es “Extraer método” que implica la modularización del mismo.

Usability refactoring

A lo largo del tiempo el campo de estudio se amplió apuntando a las necesidades de los usuarios finales del software, no solo para mejorar la calidad del código sino también la usabilidad de la aplicación web. A raíz de esto surge el concepto de “*usability refactoring”* [Garrido11] que implica hacer cambios sobre la interfaz web de una aplicación para mejorar la usabilidad manteniendo su funcionalidad. De manera similar al refactoring de código, se denominan *“usability smells”* a los problemas de usabilidad que indican la necesidad de aplicar *usability refactoring* [Grigera17a]. La idea principal es mejorar la experiencia del usuario con la aplicación y por lo tanto la usabilidad sin alterar las operaciones que este puede realizar sobre la web.

A continuación se lista el catálogo de los malos oleres de usabilidad desarrollado por Grigera et al. [Grigera17a]:

1. **No processing page (ausencia de página de proceso)**: indica que un proceso largo se está ejecutando y el usuario no tiene noción de eso ya que no existe un mensaje o algo que les indique qué está ocurriendo.

2. **Free input for limited values (campo de texto libre para valores limitados):** ocurre cuando un campo de texto libre es utilizado para ingresar un conjunto de valores limitados. Si bien las opciones son restringidas, los usuarios pueden escribir libremente y siempre es necesario que completen el texto.

3. **Scarce search results (resultados de búsqueda escasos):** ocurre cuando un formulario de búsqueda no trae los resultados como se espera.

4. **Unformatted input (campo sin formato):** ocurre cuando existe un campo de texto libre pero el usuario se ve limitado a cumplir cierto formato en los valores aceptados. Por ejemplo, si se trata de un campo que solicita ingresar una fecha, por más que te permita ingresar cualquier valor, el usuario debe limitarse a ingresar la fecha en el formato exigido para ser válida.

5. **Late validation (validación tardía):** ocurre cuando las validaciones no se muestran a tiempo al usuario luego de que este aprieta el botón *submit*, es decir, existe validaciones del lado del cliente pero se realizan recién cuando se intenta enviar el formulario al servidor. Esto puede ocasionar demoras innecesarias.

6. **No client validation (ausencia de validación en el cliente):** no existen validaciones del lado del cliente antes de que se envíe la información al servidor.

7. **Wrong default option (opción por defecto incorrecta)**: ocurre cuando no se setea el valor por defecto correcto, es decir, el más popular o el que se estima que será el que con mayor frecuencia se seleccionará en un radio button o una lista de selección. Esto provoca que el usuario pierda tiempo en seleccionar la opción correcta en casos de que los formularios sean extensos.

8. **Unresponsive element (elemento inmutable)**: ocurre cuando un usuario intentar realizar una acción haciendo click sobre algún elemento y este no responde. Esto se da porque dicho elemento provoca cierta confusión en el usuario y este piensa que presionándolo con el mouse puede llegar a realizar alguna acción.

9. **Unmatched input size (tamaño de campo inadecuado)**: ocurre cuando un campo de texto permite ingresar una cantidad de caracteres mayor o menor al tamaño que refleja. Por ejemplo si el campo es más chico de lo que en realidad permite escribir puede ser confuso para el usuario ya que solo podrá ver una pequeña parte del texto que está ingresando. También se da en el caso inverso, cuando el campo de texto es más grande de la cantidad de caracteres que permite.

10. **Undescriptive element (elemento no descriptivo)**: el usuario no comprende correctamente la funcionalidad de un botón o un link ya que este es poco auto-descriptivo, es decir, no posee un *tooltip*[2] o el nombre no es lo suficientemente representativo como para entender su funcionalidad.

[2] Tooltip: elemento gráfico en la interfaz del usuario. Se utiliza en conjunto con el cursor de mouse. Cuando el usuario pasa por arriba del elemento el mouse aparecerá el tooltip, es decir, un mensaje que describe el elemento.

11. **Overlook content (contenido ignorado)**: indica que el usuario mira por arriba el contenido de la página ya que éste es demasiado, perdiéndose así gran parte del mismo. Esto ocurre por lo general cuando el usuario ignora la mayor parte del contenido hasta llegar al punto que le interesa.

12. **Abandoned form (formulario abandonado):** se detecta cuando un usuario deja parte de un formulario sin completar, esto puede darse porque el formulario es demasiado largo y provoca que el usuario se pierda. También se da en los casos en que el formulario es demasiado complejo donde el usuario se cansa o se frustra y lo abandona.

13. **Distant content (contenido distante)**: ocurre cuando para llegar a una página destino se debe ingresar por varios links previamente. Si los usuarios solo navegan a través de los links intermedios para llegar a ese destino, sin detenerse en el contenido de cada uno, aparece este *smell*. La solución es acortar el camino para que el usuario pueda acceder con mayor rapidéz al destino.

14. **Forced bulk action (acción en lote forzada)**: hace referencia a aquellas acciones que pueden ser aplicadas a un grupo de elementos al mismo tiempo. Las secuencias de pasos previos para realizar dicha acción pueden ser confusas y tediosas si sólo se deseara aplicarla sobre un solo elemento. Este smell puede ser bastante molesto especialmente para aquellas personas que utilizan lector de pantalla para realizar las acciones, ya que implicaría constantemente ir hacia adelante (para leer el elemento) y luego volver hacia atrás (para realizar la acción).

15. **Misleading link (vínculo confuso):** ocurre cuando un usuario no entiende bien a dónde lo lleva el vínculo y para averiguarlo presiona sobre éste. O bien cuando pone el cursor arriba del link para ver la descripción del mismo (*tooltip*) pero esta no existe.

16. **Useless search results (resultados de búsqueda inútiles)**: en este caso, a diferencia del *smell scarce search results*, el formulario de búsqueda sí trae resultados pero no son los esperados por el usuario lo que provoca que este vuelva a realizar otra búsqueda o simplemente la abandone.

El catálogo anterior fue desarrollado en el contexto de una herramienta de detección automática de estos smells a partir de los eventos de interacción de los usuarios con una aplicación web [Grigera17a]. Esta herramienta tiene como objetivo ayudar a los dueños o desarrolladores de un sitio web a encontrar automáticamente problemas de usabilidad. Sin embargo, la herramienta no puede detectar todos los problemas, y genera cierto rango de falsos positivos. Junto con la detección automática de BS, la herramienta desarrollada por Grigera, llamada Kobold, permite a los desarrolladores del sitio aplicar refactorings de usabilidad para solucionar los BS detectados [Grigera17b]. Kobold puede aplicar algunos refactorings automáticamente, otros de manera semi-automática, y otros no se pueden aplicar porque necesitan intervención manual.

El catálogo completo de refactorings de usabilidad desarrollados junto al/los smell/s asociado/s es el siguiente [Grigera17b]:

1. **Add processing page:** se agrega un *overlay* a la acción que lleva tiempo procesarse, es decir, se pone la página en segundo plano y en color gris y en primer plano una leyenda indicando que se está ejecutando un proceso. Smell asociado: *No processing page.*

2. **Add autocomplete:** agrega a un campo de texto libre un autocompletado con una lista de valores posibles asociados. Smell asociado: *free input for limited values* y *scarce search results*.

3. **Add Date Picker:** se aplica cuando existe un campo de texto libre donde hay que ingresar una fecha. Smell asociado: *unformatted input*.

4. **Add form validation:** agrega validaciones a los campos de un formulario. Smell asociado: *Late validation* y *no client validation*.

5. **Date input into selects**: transforma un campo de texto libre para el ingreso de fechas por tres columnas de selección; una para el día, otra para el mes y otra para el año. Smell asociado: *unformatted input*.

6. **Turn input into radios**: reemplaza un campo de texto libre por *radio buttons* que se crean en base a una lista pequeña de valores recurrentes. Es importante que la lista de valores sea pequeña. Smell asociado: *free input for limited values*.

7. **Provide default option**: cambia el valor por defecto de la lista de selección. Smell afectado: *wrong default option*.

8. **Turno attribute into link**: convierte un elemento en un link. El valor de la página a la cual dicho link debe redireccionar deberá ser proveído. Smell asociado: *unresponsive element*.

9. **Resize input**: incrementa el tamaño del campo de texto. Se debe ingresar el tamaño que se desea. Smell asociado: *unmatched input size*.

10. **Rename element**: se renombra el elemento, también se puede agregar una descripción o un ícono, para que sea más descriptivo y el usuario pueda comprender mejor su funcionalidad. Smell asociado: *undescriptive element*.

11. **Link to top**: se agrega un link que lleva al tope de la página. Smell asociado: *overlook content*.

12. **Add tooltip**: un *tooltip* es una herramienta de ayuda visual. Se agrega el *tooltip* al elemento deseado. Se debe indicar cuál es el texto que se desea agregar. Smell asociado: *undescriptive element*.

13. **Add link**: se agrega un link al lado del link inicial. Se debe indicar la url del link destino. Smell asociado: *distant content*.

14. **Distribute menu**: distribuye las acciones globales en cada elemento de la lista para que se convierta en una acción local de este, es decir, cada elemento tendrá esa acción para poder ser ejecutada de manera individual. Smell asociado: *forced bulk action*.

15. **Split Page**: divide en varias páginas el contenido de una. Se debe indicar cuál será la página principal, su nombre y su xPath[3]. También cuáles serán las secciones en las cuales se dividirá. Smell asociado: *abandoned form* y *overlooked content*.

Client-side web refactoring (CSWR)

La técnica CSWR se utiliza para realizar mejoras a un sitio web sin modificar su funcionalidad [Garrido13a]. El objetivo principal de estos refactorings es el de incrementar la usabilidad y accesibilidad del sitio de forma que cualquier persona, sin importar su condición o discapacidad, puede utilizarlo sin problemas.

CSWR propone realizar mejoras sobre la estructura o la apariencia del sitio web sin modificar su contenido ni sus funcionalidades eliminando de esta forma los *bad smells* de accesibilidad y usabilidad. Propone ejecutar los refactorings del lado del cliente sobre una aplicación ya desplegada en producción sin necesidad de modificar el servidor, ya que esto requeriría del contacto con la empresa desarrolladora del software y no sería tan fácil de aplicar.

[3] xPath: puede ser utilizado para navegar entre elementos y atributos en un documento xml (incluye documentos HTML). Cada expresión xPath corresponde a un único elemento o a un único conjunto de elementos. Ej: /html/body/div.

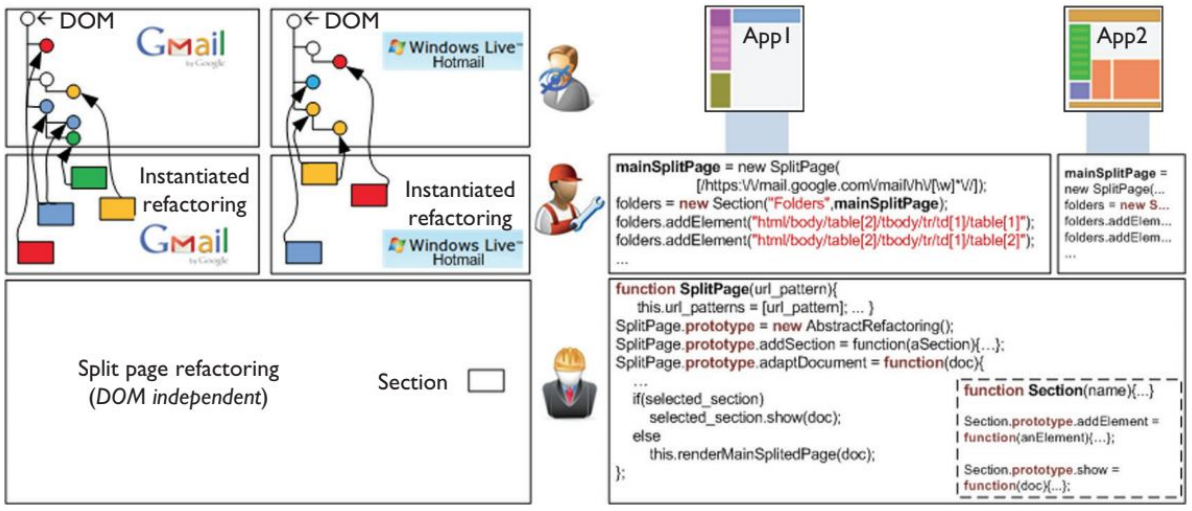


Figura 2.2.1. Esta figura corresponde al trabajo realizado sobre CSWR [Garrido13a]. Se muestra el comportamiento y manejo del DOM donde se aplica el refactoring solo alterando el lado de cliente, en el ejemplo, Split Page.

Esta propuesta se basa en dos puntos importantes:

- Mantenimiento simple: los desarrolladores del sitio realizan refactorizaciones en el sitio que son dirigidas a los usuarios en general, mientras que CSWR apunta a la posibilidad de realizar varios refactorings apuntando a las necesidades de diferentes grupos de usuarios.
- Independiente de la arquitectura: no es necesario conocer en profundidad la arquitectura del sitio, solo saber lo necesario para poder adaptar el refactoring necesario en el cliente.

La figura 2.2.1 muestra los tres niveles que involucra el proceso de refactoring:

1) Desarrollo de un programa Javascript que utilizará componentes del sitio y de la herramienta provista para realizar las modificaciones.
2) El nivel intermedio representa el punto donde el desarrollador puede utilizar los refactorings instanciados provistos por la herramienta o bien desarrollar los propios refactorings instanciados.
3) Por último el usuario final selecciona los refactorings que desea instalar en su navegador. Esto permite que el usuario pueda configurar los refactorings según sus necesidades.

El trabajo realizado en *Personalized web accesibility* [Garrido13a], presenta un *framework* desarrollado con XUL/XPCOM overlays, extensión de navegador Firefox. Permite cargar refactorings asociados a determinados sitios webs y ejecutarlos en tiempo real. A estos refactorings se les da un orden de ejecución, de modo que no se sobreescriban y se pueda evitar cualquier comportamiento no deseado. También debe especificarse la URL y el elemento o elementos donde se aplicará dicho refactoring en concepto de xPath ya que de otro modo podría provocar modificaciones en sitios que no se quería realizar dicha

modificación. Es por eso que también deberá tenerse en cuenta los refactorings que se instanciarán primero a la hora de especificar el xPath del siguiente ya que este podría modificarse debido a la ejecución de otro refactoring previo. Finalmente cada vez que se ingrese al sitio correspondiente a cada refactoring éste se ejecutará según corresponda.

Con el rápido avance que ha tenido la tecnología web en los últimos años este framework de CSWR se fue desactualizando y cayó en desuso. En esta tesina se desarrolló una nueva herramienta basándose en los conceptos utilizados por el framework mencionado pero combinando tecnologías modernas como Django [Django05] y API WebExtension como se describe más adelante.

Automatic detection of usability smells in web applications

"Kobold" es una herramienta que funciona como un servicio web y tiene dos funcionalidades importantes: por un lado, la detección automática de problemas de usabilidad (que a partir de la jerga de refactoring se denominan *"usability smells*") a partir de logs de eventos de interacción, y por otro lado permite aplicar CSWR como soluciones a cada uno de los *bad smells* encontrados [Grigera17]. Las soluciones se generan automáticamente, aunque a veces es necesaria la intervención de un usuario para ingresar algunos valores requeridos para la generación de los refactorings.

La desventaja de esta aplicación es que los reportes deben ser leídos por técnicos y existen algunos reportes que no pueden ser solucionados debido a que aún no existen refactorings desarrollados para este fin.

2.3 Crowdsourcing

En los últimos años el incremento de los sitios web ha sido excesivo. En consecuencia también han incrementado las exigencias de los usuarios. Esto provoca muchas veces que las compañías deban atender las necesidades de estos de forma rápida y eficáz. El crowdsourcing incrementa exponencialmente la probabilidad de encontrar una solución rápida y eficiente, ya que muchos de los voluntarios involucrados son usuarios del sistema y por lo tanto, quién mejor que ellos para saber qué es lo que el sitio necesita para mejorar [Howe08].

Crowdsourcing [Brabham14] es un nuevo modelo de negocio en línea que utilizan las compañías para solucionar un problema a través de la colaboración de varios usuarios. La idea principal para motivar a los usuarios era la de premiar la mejor solución de manera que ésta pueda ser utilizada como propia por la empresa.

El crowdsourcing consiste en exponer un problema a la comunidad que utiliza el sitio web y permitirles contribuir con posibles soluciones. La mejor de todas será la elegida. Una de las razones principales por las cuales se elige este método para la resolución de problemas es que un grupo de personas trabaja mejor que una sola. Se presentarán

muchas soluciones posibles y se elegirá la que mejor se adapte a las necesidades del problema.

Crowdsourcing para mejorar la usabilidad

Es posible mejorar la usabilidad de las aplicaciones web a través del crowdsourcing y esto se da gracias al aporte de los usuarios. Adicionalmente se puede destacar que tendrá un costo más bajo para la empresa y le permitirá obtener ideas nuevas ya que los usuarios voluntarios serán los responsables de resolver los problemas de usabilidad. Esto ayuda a conocer mejor a los usuarios de sitio y mejorar la comunicación con ellos y también satisfacer sus necesidades de la mejor forma. En consecuencia la empresa no sólo mejorará la usabilidad del sitio, sino que también sus ingresos.

Consideramos que la mejor forma de mejorar la usabilidad de una aplicación web es dejar que el usuario de esta tome la decisión de cómo debería verse y que también realice o proponga las posibles mejoras en el sitio ya que éste al utilizar la aplicación con frecuencia es el que más interesado está en realizar ese tipo de mejoras.

Herramienta social para la mejora de accesibilidad

La herramienta Social Accessibility [Takagi08] propone mejorar la accesibilidad de las aplicaciones web utilizando el potencial de la comunidad. El usuario tiene la posibilidad de reportar el problema utilizando la herramienta *Screen Reader* ofrecida por el equipo y un grupo de técnicos voluntarios discutirán la mejor solución posible a este y generarán una serie de metadata para solucionarlo. Luego dicha metadata será publicada para poder ser accesible por cualquier usuario que enfrente el mismo problema. La herramienta está orientada para usuarios con disminución visual.

El objetivo principal es aumentar la accesibilidad del sitio web reportado adaptándolo de tal forma que mejore la navegación de la aplicación a través de la utilización de los lectores de pantalla. Una vez que los usuarios realizaron los reportes, estos estarán inmediatamente a disposición de los voluntarios encargados de resolverlos. Luego de realizar un análisis exhaustivo y haber determinado cuál sería la mejor solución, los expertos crearán metadata que podrá ser aplicada en la web para cumplir con el objetivo de mejorar la accesibilidad. El método que utilizan para aplicar la metadata es el de transcodificación[4].

En la herramienta Social Accessibility la transcodificación es aplicada para mejorar la accesibilidad mediante la modificación del DOM. La modificación se realiza en el momento y no es necesario instalar ninguna aplicación extra. Esta codificación podría generar pérdida de datos por lo cual deberá usarse con cuidado. La aplicación trata de recolectar todos los datos necesarios para ser lo más preciso posible.

[4] Transcodificación: conversión directa de un codificador a otro. Podría existir la posibilidad de tener pérdida de datos.

Una de las limitaciones más significativas de Social Accessibility es que se necesitan voluntarios expertos en el área que entiendan sobre metadata y accesibilidad para la creación y aplicación de estas y a su vez puedan aprender a utilizar la herramienta fácilmente. Esto reduce considerablemente la comunidad de voluntarios creando limitaciones y retrasos importantes en la resolución de los reportes.

2.4 Tecnología de Extensiones para un navegador web

Un navegador o explorador web (browser en inglés) es un programa informático que permite a los usuarios acceder a los sitios web deseados a través de la dirección URL conocida. Se comunica a través de internet con el servidor donde se encuentra alojada la información requerida e interpreta la respuesta para luego mostrarla al usuario de forma que éste pueda entenderla.

Normalmente vienen ya instalados con el sistema operativo o también se pueden descargar de internet e instalarse. Los navegadores más populares en la actualidad son Google Chrome, Internet Explorer y Mozilla FireFox.

Extensiones de navegador

Son programas informáticos que se instalan en el navegador y agregan comportamiento al mismo.

WebExtension

Las APIs WebExtesion [MDN18] son utilizadas para construir aplicaciones para los navegadores Firefox. Cumplen los estándares de tecnología web de HTML, CSS, Javascript. Las soluciones desarrolladas con WebExtension son compatibles con Google Chrome y Opera es por ello que decimos que es un sistema Cross-Browser.

Las extensiones son desarrolladas para satisfacer las necesidades del usuario, se comunican con las páginas web para cambiar su apariencia o agregar comportamiento. También pueden comunicarse con el navegador para ampliar sus características.

Configuración

La configuración de las extensiones se realiza a través del archivo manifest.json. Algunos de los parámetros más importantes de dicho archivo son:

- "Manifest_version" : identifica el número de versión de la extensión. Parámetro obligatorio.
- "Name" : nombre de la extensión. Parámetro obligatorio.
- "Version" : versión correspondiente. Parámetro obligatorio.

- "Description" : descripción de la extensión.
- "Icons" : ícono de la extensión que aparecerá en el administrador. Se pueden utilizar íconos para diferentes resoluciones de pantalla.
- "Content_scripts" : El parámetro más importante ya que permite ejecutar scripts del lado del cliente siempre y cuando matchee con las URLs especificadas.

Content scripts

Como se mencionó anteriormente los content scripts son claves en la extensión para la comunicación con el contexto de la página web. Estos scripts pueden manipular y alterar el DOM, sin embargo no pueden acceder a las variables javascript definidas en los scripts de la página web y de la misma forma los scripts de la página web no pueden acceder a las variables javascript de los content scripts. Los content scripts tampoco tienen acceso a los scripts cargados por la página web, es decir, si se utiliza alguna librería extra en la página web, los content scripts no pueden utilizarla, si así quisieran, deberían agregar la librería a través del parámetro content_scripts, con la clave "js" en el archivo manifest.json tal como se ve en el siguiente ejemplo:

```
"content_scripts": [ { "matches": ["*://*.mozilla.org/*"],
                       "js": ["jquery.js", "content-script.js"] } ]
```

Los contents scripts solo puede acceder a un pequeño conjunto de funcionalidades de WebExtension Javascript APIs pero son suficientes como para permitir una comunicación fluída entre el contexto de la página web y el contexto de la extensión (content scripts y background scripts) lo cual les permite acceder indirectamente a todas las funcionalidades de WebExtension Javascript API.

Background scripts

Son scripts que son parte de la extensión en sí. Estos pueden acceder a todo el contenido dentro de WebExtension Javascript APIs, pero no pueden acceder directamente al contenido de las páginas web.

Prueba y publicación de la extensión

Para probar la extensión:

- En el navegador firefox escribe **about:debugging** y luego hacer click en el botón "Cargar complemento temporal" y seleccionar cualquier archivo de la extensión.
- También se puede levantar la extensión utilizando el comando **web-ext run** dentro de la carpeta donde se encuentra el código fuente de la extensión.

En ambos casos cuando se reinicia el navegador firefox se desinstalará la extensión.

Para publicar la extensión al público general se deberá enviar al equipo de Firefox el cual deberá aprobarla y firmarla para que pueda ser vista y descargada por el resto de los usuarios de Firefox.

2.5 Conclusión

CSWR es un framework que ofrece la posibilidad de aplicar refactorings de usabilidad del lado del cliente. El desarrollo de los refactorings deberá realizarse por expertos informáticos y deberán instalarse con cuidado en el orden correcto para no alterar su funcionalidad. El desarrollo del framework se realizó utilizando XUL/XPCOM overlays lo cual es un problema en la actualidad ya que se encuentra discontinuado, provocando que la herramienta sea obsoleta.

Social Accessibility ofrece la posibilidad de reportar problemas de accesibilidad de un determinado sitio web. Estos problemas están orientados a usuarios que utilizan lectores de pantalla. Una vez realizado el reporte éste se encontrará disponible de forma inmediata para que los voluntarios puedan discutir y desarrollar su solución en concepto de metadata. Una vez desarrolladas las metadatas se publicarán para que puedan ser utilizadas por todos los usuarios interesados.

Este trabajo propone la utilización de ambos enfoques para desarrollar un conjunto de herramientas que ofrecen la posibilidad de reportar problemas de usabilidad en cualquier aplicación web y posteriormente buscar una solución en términos de CSWRs. Los voluntarios podrán elegir los refactorings que deseen instalar en sus navegadores y la ejecución de los refactorings se realizará del lado del cliente.

Se propone ampliar el campo de reporte a cualquier tipo de *usability smell*, no solo cuestiones de accesibilidad. También tendrá la ventaja de que los usuarios no deberán ser expertos informáticos ya que las herramientas serán intuitivas y descriptivas de tal forma que cualquier tipo de usuario podrá hacer uso de estas.

Finalmente en este trabajo se aplica el concepto de crowdsourcing permitiendo a la comunidad participar abiertamente de los reportes y de las resoluciones de los mismos. Para lograr esto se agregan funcionalidades que permiten a los usuarios reportar malos olores de usabilidad y al mismo tiempo solucionarlos mediante la interacción con una interfaz gráfica donde se explicará claramente el procedimiento haciéndolo más fácil y amigable.

CAPÍTULO 3

Arquitectura de la plataforma colaborativa de gestión de usabilidad

3.1 Introducción a Mellito

La usabilidad y accesibilidad son características muy importantes de las aplicaciones web que en general no son tenidas en cuenta por el alto costo que implica su evaluación y reparación, sobre todo para las pequeñas y medianas empresas. Cabe destacar que, si se realizaran las inversiones para mejorar la usabilidad, estas adaptaciones no les garantiza satisfacer a todos los usuarios.

Debido al incremento en la utilización de las aplicaciones web con fines comerciales es de gran importancia para las empresas enfocarse en estos aspectos de usabilidad ya que sino la carencia de entendimiento por parte del usuario podría provocar pérdidas en la compañía.

Mellito, nace por la falta de determinación por parte de las compañías para priorizar la usabilidad en cara a las aplicaciones web. Su principal objetivo es el de satisfacer las necesidades del cliente enfocándose en mejorar la usabilidad en las aplicaciones web que estos consumen. Para lograr esto se ofrece la posibilidad de que la comunidad pueda aportar soluciones a estos problemas y el usuario final pueda utilizarlas para mejorar su experiencia en estas aplicaciones.

Mellito, está compuesto por un conjunto de herramientas desarrolladas para la mejora de usabilidad en aplicaciones web a través de la resolución de usability smells. Está compuesta por tres herramientas: Usability smell reporter, Usability smell manager y Usability smell fixer. Todas estas trabajan en conjunto para cumplir con el objetivo de mejorar la experiencia del usuario en el sitio web y ayudar a que las aplicaciones se adapten a las necesidades del usuario final ofreciéndole soluciones para aplicar de manera local.

La ventaja de estas herramientas es que para realizar la adaptación no es necesario esperar a que las modificaciones se realicen del lado del servidor por los desarrolladores de la aplicación web ya que los cambios se aplicarán del lado del cliente. También cabe destacar que las modificaciones que el usuario realiza sobre la aplicación se adaptan a sus necesidades, es decir, corresponden a lo que éste desea para mejorar su experiencia en el sitio. Sin embargo, si el desarrollador del sistema realiza modificaciones de usabilidad del lado del servidor, éstas no nos aseguran que son las que todos los usuarios finales necesitan o desean.

Uno de los principales enfoques de Mellito es el de ampliar el campo de voluntarios y darle la oportunidad a toda persona que quiera participar en los reportes o soluciones de estos sin necesidad de que sea técnico informático especializado en el área ya que las herramientas son bastante intuitivas y descriptivas. Cada una de estas posee descripciones e indicaciones en cada paso a realizar. Es importante que el usuario se sienta cómodo de

usarlas y entienda bien cada concepto para que pueda encontrar de forma fácil y precisa el reporte de un *usability smell* y la resolución del mismo.

3.2 ¿Qué ofrece Mellito?

Mellito permite realizar reportes de *usability smells* y también gestionarlos para su posterior resolución mediante la ejecución de *usability refactorings* del lado del cliente.

A la hora de reportar un *usability smell* la herramienta ofrece dinamismo al tratarse de una extensión de navegador dándole la oportunidad al usuario de navegar normalmente internet y, en el momento que crea conveniente, realizar el reporte con solo desplegar la extensión. Es decir, no es necesario salir del sitio y abrir una aplicación distinta tal como puede verse en la figura 3.2.1.

La primera herramienta que se presenta dentro del flujo de reporte y resolución de *usability smells* es *Usability smell reporter*. Cuando el usuario elige la página web candidata, abre la extensión de navegador y comienza a completar los datos que ésta le solicita.

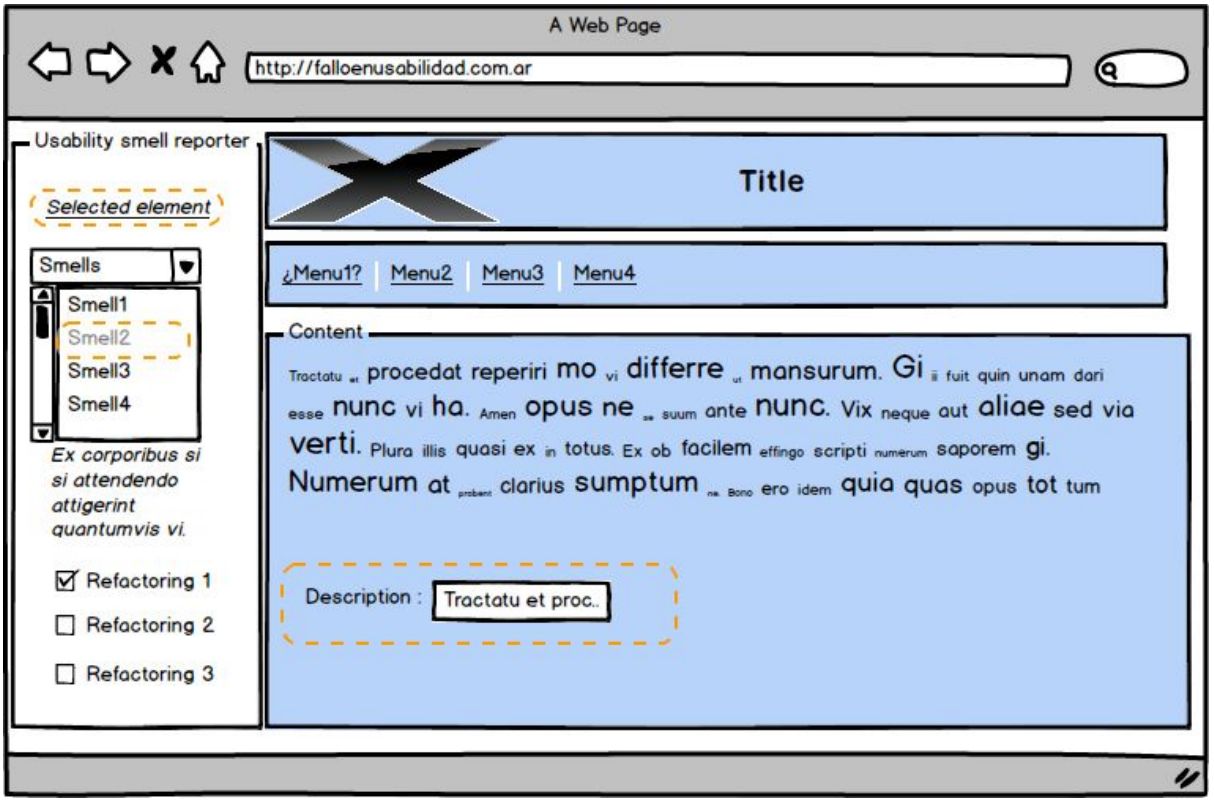


Figura 3.2.1. Ejemplo de cómo identifica el usuario un *usability smell* en el sitio web y lo asocia a un determinado *smell*. Se puede observar en el margen izquierdo la extensión desplegada y el sitio web del lado derecho perfectamente visible para realizar el reporte de forma cómoda y ágil.

Es muy fácil realizar los reportes ya que se ofrece una interfaz visual muy amigable donde el usuario será introducido y guiado durante todo el proceso para clasificar de la forma más exacta posible los problemas de usabilidad existentes en la aplicación web. Asimismo en la figura 3.2.1 también puede observarse que la extensión ofrece un listado de refactorings a aplicar para la URL actual. Estos refactorings son producto de los reportes realizados y posteriormente solucionados. El usuario tendrá la posibilidad de ejecutarlos y guardarlos de manera local o simplemente no utilizarlos.

Una vez que el usuario termina de realizar el reporte aparecerá en juego la segunda herramienta involucrada en el flujo, *Usability smell manager.* En esta se podrá seleccionar el reporte para plantear una solución. Cuando se selecciona el reporte, la aplicación llevará al usuario al sitio correspondiente y éste deberá abrir la extensión *Usability smell fixer,* última herramienta del flujo, para continuar con el siguiente paso, buscar una solución al problema. En este caso tampoco será necesario ser un experto en el área ya que la aplicación permitirá al usuario probar los refactorings disponibles en el mismo sitio web afectado para poder determinar con mayor precisión cuál es el ideal para solucionar el reporte (Figura 3.2.2). La interfaz gráfica que se ofrece es intuitiva y fácil de seguir para cumplir este objetivo.

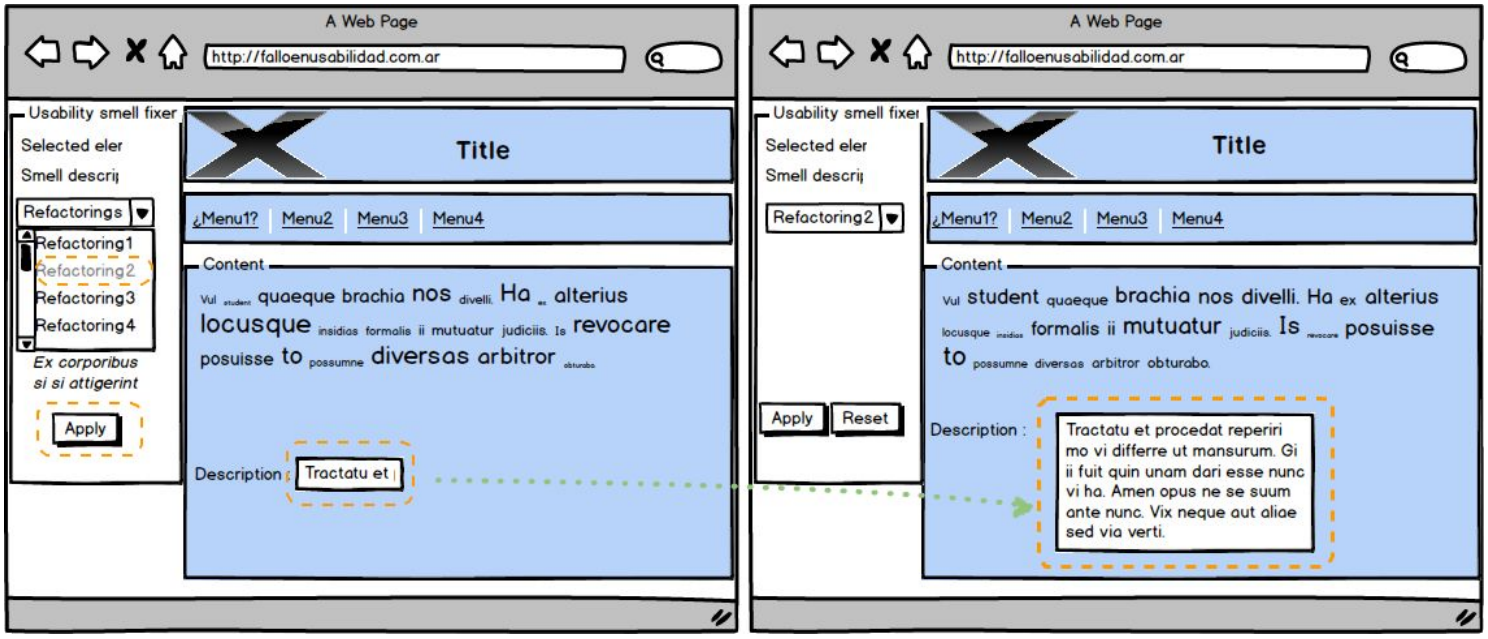


Figura 3.2.2. En esta figura se muestra cómo *Usability smell Fixer* permite ejecutar los *refactorings* y probarlos para analizar si estos producen el resultado esperado. Se podrá probar el listado completo de refactorings presentado ya que éstos serán todos los que estarán relacionados al *usability smell* reportado.

Las extensiones se encuentran comunicadas constantemente con la herramienta Kobold [Grigera17b]. Como se describió en el Capítulo 2, Kobold es una herramienta que se construyó con el objetivo de ayudar al dueño de un sitio web. En el contexto de este trabajo se utiliza Kobold como servicio que provee los refactorings que solucionan un BS. A la hora de solicitarle la lista de *usability smells* o la lista de *usability refactorings*, Kobold es el encargado de enviarlas. De esta forma esta información se encuentra centralizada y en el caso de cualquier tipo de modificación o actualización se verá reflejada constantemente.

Para lograr la comunicación entre ambas aplicaciones se desarrolló en este trabajo, conjuntamente con Kobold, una API. Esta se encuentra del lado de Kobold y expuesta para ser consumida por las aplicaciones que componen Mellito.

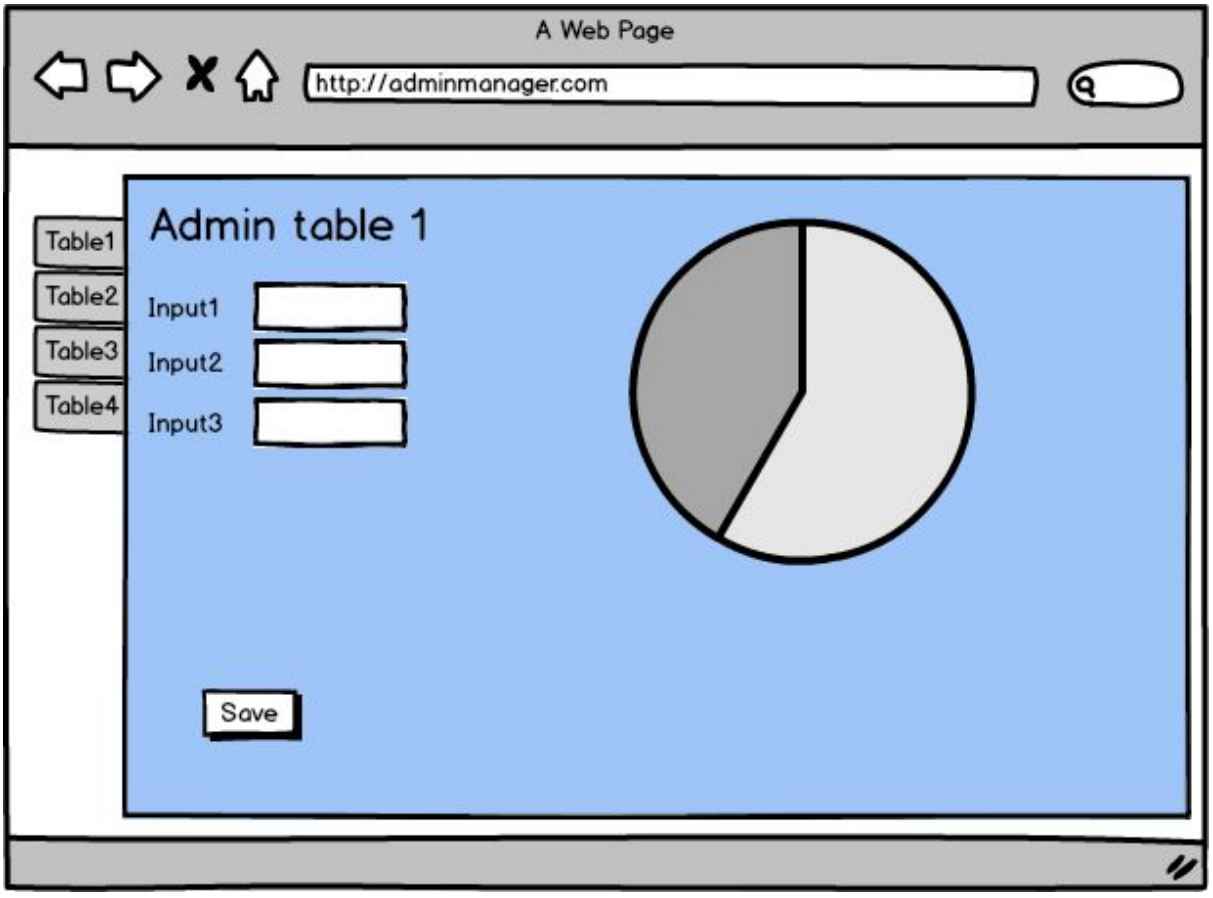


Figura 3.2.3. Admin manager. Administrador de tablas de la base de datos SQLite3. Esta funcionalidad se encuentra presente en la herramienta *Usability smell Manager*.

Usability smell manager ofrece una interfaz para administrar la base de datos centralizada (Figura 3.2.3). Esto permite mantener consistentes los datos utilizados por la tres aplicaciones. La interfaz es fácil e intuitiva para un técnico novato, sin embargo, éste debe tener conocimientos técnicos básicos sobre administración de bases de datos para administrar de la mejor forma posible la información. En esta base de datos se guardarán todos los reportes realizados y la información que involucra su resolución.

Diagrama de arquitectura

En el diagrama presentado a continuación (Figura 3.2.4) se puede ver cómo se interrelacionan todas las aplicaciones que forman parte del flujo completo de reporte y resolución de *usability smell* junto con las tecnologías utilizadas para el desarrollo de cada componente.

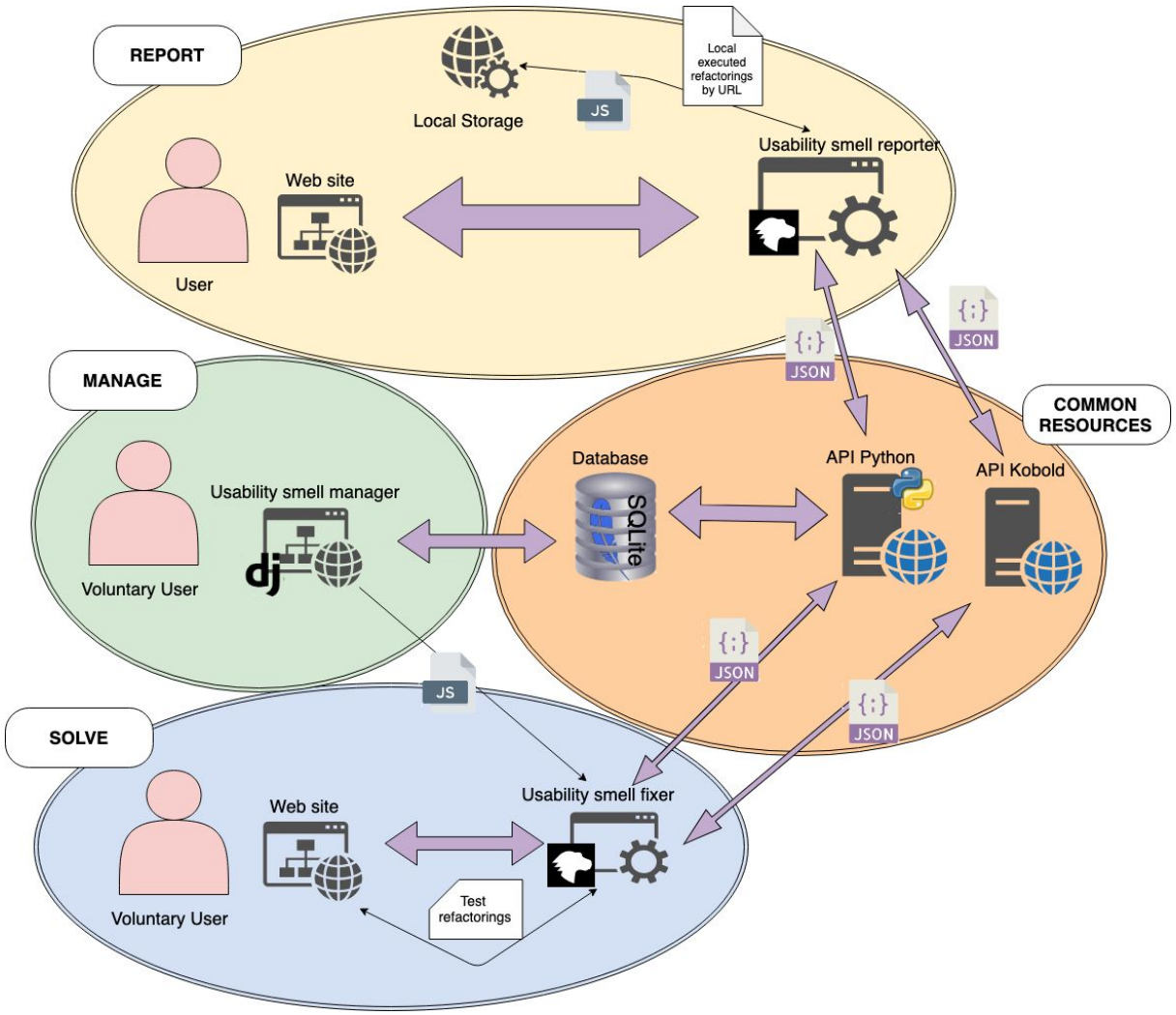


Figura 3.2.4. Diagrama de arquitectura. Se presenta la arquitectura utilizada para el desarrollo de Mellito y la interrelación entre todas las aplicaciones que lo componen.

Descripción de los componentes

El diagrama se divide en cuatro denotadas secciones para definir los componentes involucrados y cómo se relacionan entre ellos. A continuación se explica cada una de ellas:

1. **Sección de componentes de reporte**: la extensión *Usability Smell Reporter* se comunica con la aplicación web intercambiando la información necesaria para la generación de los reportes de problemas de usabilidad, es decir, reportes de BS. Asimismo, la extensión se encarga de aplicar los CSWRs

disponibles para el sitio visitado. Para esto solicita al *local storage*[5] los *refactorings* ejecutados en el cliente y a la API Python aquellos *refactorings* disponibles para ejecutar en esa página web específica. La comunicación con la API Kobold se realiza para obtener el listado de smells necesario para la generación del reporte y el código de los refactorings disponibles.

2. **Sección de componentes de gestión:** la aplicación web *Usability smell manager* actúa como intermediario para proveer la información necesaria sobre el reporte a la extensión *Usability smell fixer*. A su vez presenta una interfaz para la administración de la base de datos SQLite3.

3. **Sección de componentes de resolución:** la extensión *Usability smell fixer* recibe información del componente de gestión y se comunica constantemente con la API Python para recuperar y guardar datos sobre los reportes. También posee comunicación con Kobold a través de su API para la recuperación información y código Javascript de los *refactorings*. Desde la web se prueban constantemente los refactorings hasta encontrar el que mejor se adapte.

4. **Sección de componentes de recursos en común:** esta sección representa todos los recursos compartidos entre las tres aplicaciones. Las APIs se comunican a través de *requests* y *responses* JSON. La base de datos SQLite3 fue desarrollada con el framework Django y se administra a través de un administrador web desde *Usability smell manager*.

Cuando un usuario decide reportar un problema de usabilidad abre la aplicación *Usability Smell Reporter.* Esta aplicación podría o no ofrecerle algún refactoring ya disponible para esa URL. En el caso de que no posea sugerencias o, si las posee, no le agraden, el usuario tiene la posibilidad de abrir un reporte. Elige entre la lista de *usability smells* el que más se ajuste al problema encontrado y guarda el reporte.

El voluntario decide entrar a *Usability Smell Manager* para ver los listados de reportes realizados por los usuarios y elige uno para resolver. Luego de hacer *click* sobre uno de ellos abre la extensión *Usability Smell Fixer* para comenzar a resolverlo. La extensión le ofrece un listado de *refactorings* como sugerencia a la solución. El voluntario prueba los *refactorings* y elige el que considera mejor para solucionar el problema. Una vez que el voluntario guarda la solución, ésta se asocia al reporte e inmediatamente estará disponible para los usuarios de la extensión *Usability Smell Reporter* para poder ser utilizado. Ahora cuando el usuario que realizó el reporte ingrese a la extensión podrá ver la solución a su reporte en el listado y haciendo *click* en el *checkbox* ejecutará el refactoring y lo guardará en el almacenamiento local para ser ejecutado siempre que ingrese al sitio web.

[5] Local storage (almacenamiento local): tipo de almacenamiento de un navegador web. Se pueden guardar los datos sin tiempo de expiración, aún cuando el navegador es cerrado estos seguirán vigentes.

Modelo de base de datos

A continuación se presenta el modelo de base de datos que forma parte de este proceso. Este modelo se encuentra compartido por las tres aplicaciones las cuales poseen permisos para realizar modificaciones constantes sobre éste.

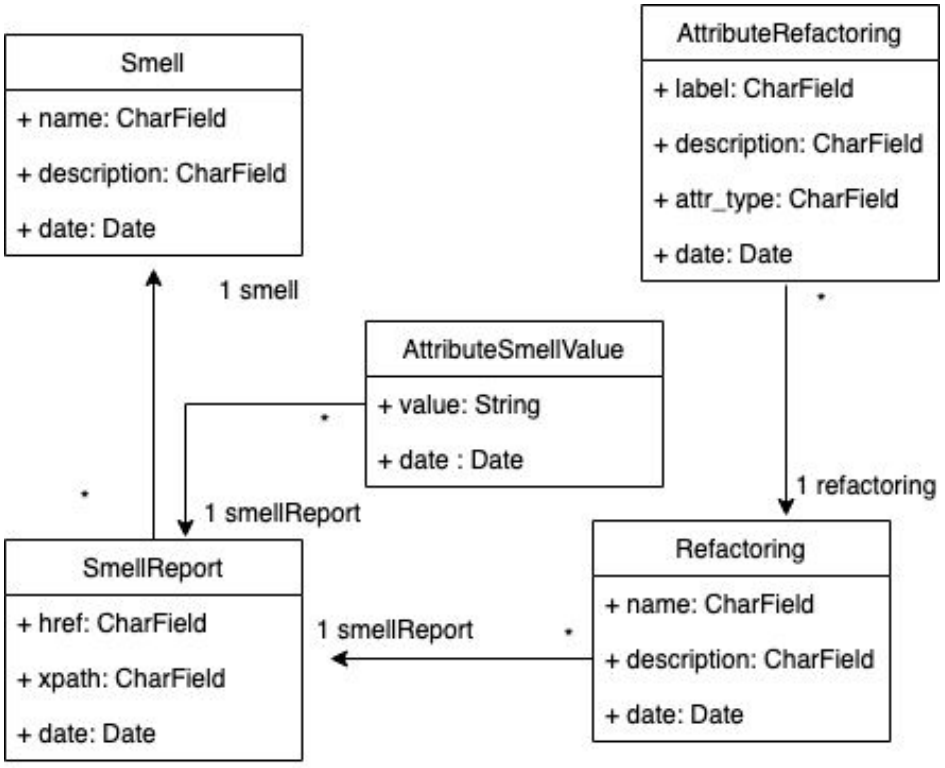


Figura 3.2.5. Diagrama de clases del modelo de base de datos.

En la figura 3.2.5 se presenta el modelo de base de datos a través de un diagrama de clases[6]. El diseño se adapta para satisfacer las necesidades de las tres aplicaciones que se utilizan en el flujo de reporte y resolución de *usability smells* de las aplicaciones web.

Se utilizará la clase **`SmellReport`** como entidad principal para almacenar toda la información necesaria sobre el reporte realizado a través de la aplicación *Usability smell reporter*. A esta entidad se le asociará el **`Smell`** correspondiente y, en el caso de que el usuario haya completado los valores de los atributos, también se guardarán en la tabla **`AttributeSmellValue`** para luego ser enviados a *Kobold* como información extra para la recuperación de los refactorings que mejor se adapten al problema. Los refactorings que se recuperan y se eligen para resolver un reporte se guardarán en la tabla **`Refactoring`**, en la cual figurará el ***idKobold*** que es el id que se utilizará para pedir a *Kobold* el refactoring directamente sin necesidad de realizar la solicitud a través del *smell* y sus atributos. La tabla **`AttributeRefactoring`** se utilizará como complemento de información obligatorio en el caso de que el refactoring necesite datos extra. Estos campos serán completados por el

[6] Diagrama de clases: lenguaje que describe la estructura de un sistema representando sus clases, atributos, operaciones y las relaciones entre los objetos que involucran el diagrama.

usuario desde la aplicación *Usability smell fixer* y se guardarán en la base de datos una vez que se asocie el refactoring como solución al reporte.

3.3 APIs

La *API Usability Smell* fue desarrollada con el framework Django [Django05] para interactuar con las aplicaciones *Usability smell manager, Usability Smell Report Manager y Usability smell fixer*. A través de esta se puede solicitar *smells*, *refactorings* y reportes realizados por otros usuarios con la extensión *Usability smell reporter*. También se puede guardar las soluciones propuestas a los reportes que se realizan a través de *Usability smell fixer* y de esta manera ofrecer al usuario el listado de soluciones para que aplique en su navegador en caso de desearlo. Otra de las funcionalidades que ofrece es la de recuperar los refactorings guardados por URL, es decir, por sitio web asociado al reporte realizado.

En el Anexo A de este trabajo se encontrará el detalle de la API para su integración.

La *API Kobold* fue desarrollada en conjunto con el equipo de Kobold para este trabajo con el fin de comunicar ambas herramientas y de esta forma permitir a Mellito solicitar la información necesaria para realizar el reporte y su posterior solución. La funcionalidad principal es la de proveer el listado de *usability smells*, para clasificar los reportes, y el listado de *usability refactorings*, para solucionar dichos reportes.

En el Anexo B de este trabajo se encontrará el detalle de la API para su integración.

CAPÍTULO 4

Usability Smell Reporter

4.1 Introducción

Usability smell reporter es una herramienta colaborativa libre y gratuita desarrollada con API WebExtension. Surge a raíz de la necesidad de tener una herramienta que pueda ser utilizada por personas expertas e inexpertas en el área de la informática. A través de esta herramienta se podrán realizar reportes que reflejen la falla de usabilidad en los sitios webs.

Si bien existen otras herramientas que ayudan a mejorar la usabilidad de una aplicación web, tales como *Kobold* [Grigera17], ninguna ofrece la posibilidad al usuario de reportar alguna deficiencia de usabilidad que puede haber detectado en uno o varios sitios webs que frecuenta a la vez. Usability Smell Reporter ofrece esta funcionalidad, cubriendo la deficiencia existente en otras herramientas. A través de la plataforma Usability Smell Reporter cualquier voluntario, experto informático o no, que identifique una falencia de usabilidad podrá generar un reporte para que eventualmente éste pueda ser solucionado.

En la actualidad los usuarios son cada vez más exigentes y ya no se conforman con los sitios convencionales, es decir, si un sitio web no les agrada o no les es cómodo existe el riesgo de que lo abandonen y buscar una alternativa, pues saben que seguramente podrán encontrar la información que necesitan en otra parte. Esto se debe a la alta demanda y competencia que existe en el mercado web.

Es por ello que vemos la necesidad de mejorar la calidad de los sitios webs y una buena forma para que esto suceda es la de incrementar la calidad del sitio en concepto de usabilidad web. Sabemos que el mejor crítico es el usuario que lo frecuenta y, seguramente, uno de los más interesados en mejorar su navegación y hacerla lo más amigable posible. A raíz de esto creamos la herramienta *Usability Smell Reporter* para que puedan realizar los reportes y estos puedan tenerse en cuenta a futuro para mejorar la usabilidad mediante la ejecución de refactorings del lado del cliente (CSWR). Cada reporte tendrá asociado un *smell,* la dirección del sitio web y la ubicación del componente afectado expresado en concepto de xPath. Esta herramienta también ofrece la posibilidad de utilizar reportes ya solucionados, es decir, que poseen refactorings asociados para revolver el problema ya que algún voluntario se encargó de confeccionar una solución para dicho reporte.

4.2.¿Cómo se utiliza?

Usability Smell Reporter deberá instalarse como una extensión de navegador en Firefox. Una vez instalada ya estará disponible para ser utilizada en el margen derecho superior del navegador tal como se ilustra en la figura 4.2.1. Para comenzar a gozar de los beneficios el usuario deberá antes elegir el sitio web que considera que posee un problema

de usabilidad para reportar y que requiere ser solucionado para mejorar la experiencia del usuario en la página.

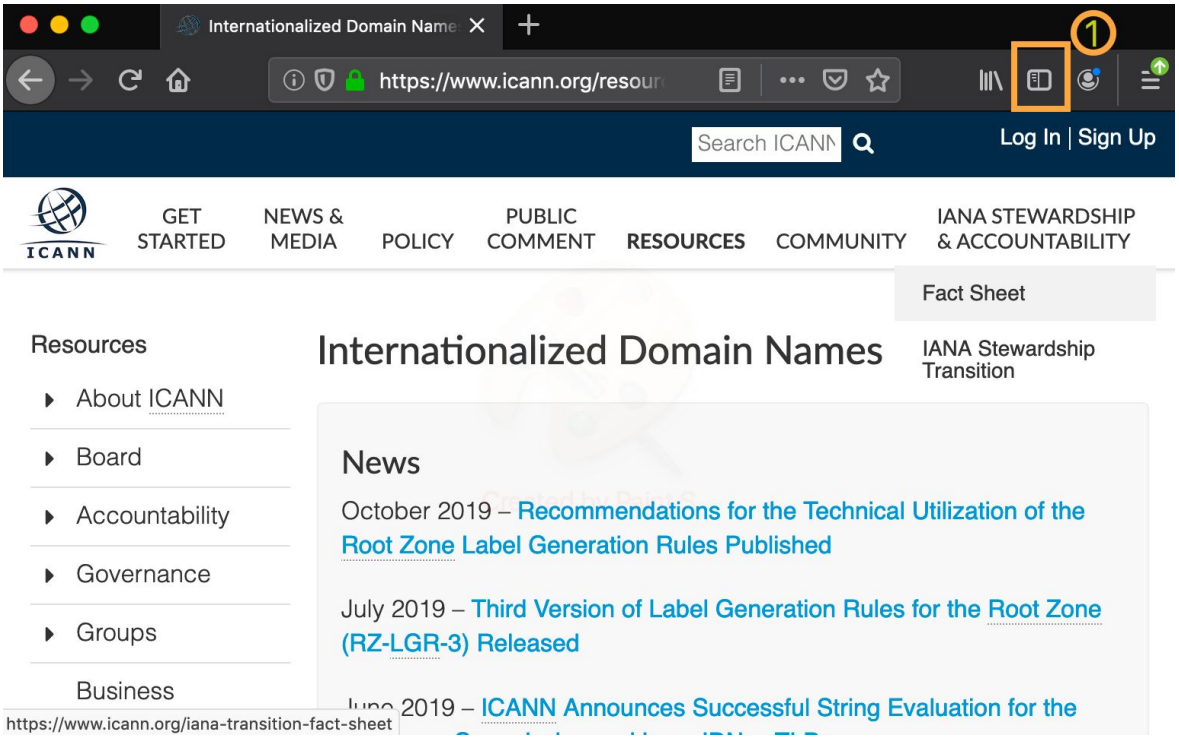


Figura 4.2.1. Representa la ubicación del botón para abrir la extensión.

Cuando finalmente el usuario elige el sitio web habilitará el reporte utilizando el botón *"enable report"* tal como se ve en la figura 4.2.2.

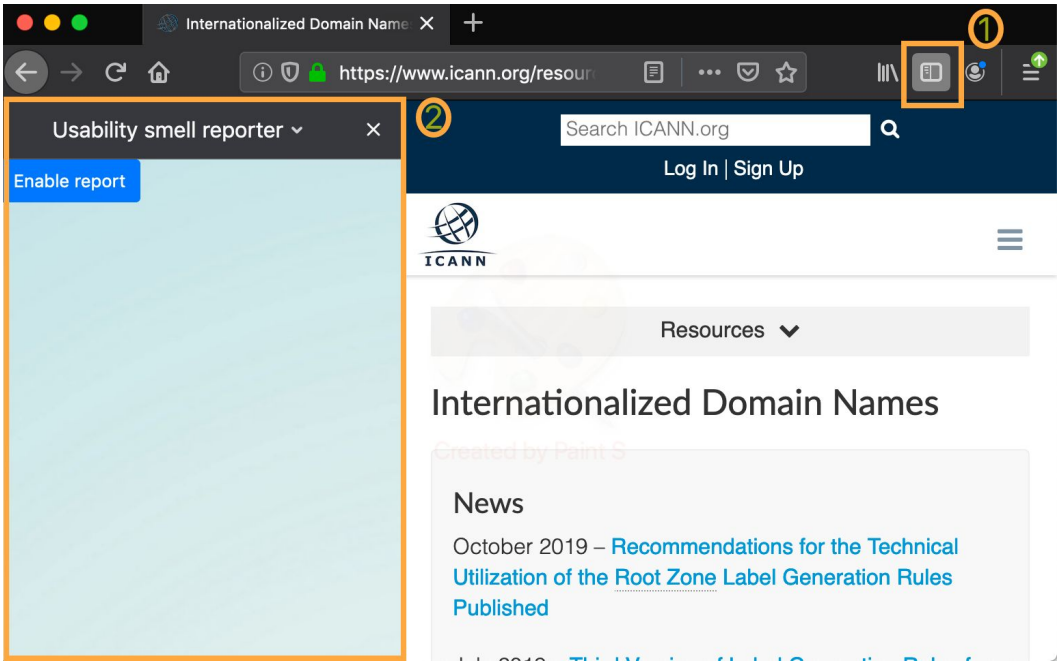


Figura 4.2.2. Inicio de la aplicación "*Usability smell reporter*"

Cuando se habilita la generación del reporte, todos los eventos que pueden llegar a realizarse en la página seleccionada se anularán y no podrán llevarse a cabo, es decir, si se desea navegar la página haciendo click en algún link éste no funcionará. Esto es para evitar cualquier tipo de funcionalidad involuntaria a la hora de seleccionar los elementos afectados. En caso de que el usuario desee cambiar de página deberá presionar el botón "disable report" para poder navegar normalmente.

Luego de habilitar la aplicación para realizar el reporte, se podrá seleccionar en la página los elementos que formarán parte del mismo. La aplicación ofrece ayuda visual para seleccionar la sección afectada correctamente. Cada vez que el voluntario pase el mouse por arriba de un elemento éste se marcará color naranja (Figura 4.2.3) para que pueda identificarlo correctamente y abarcar con exactitud el o los elementos deseados.

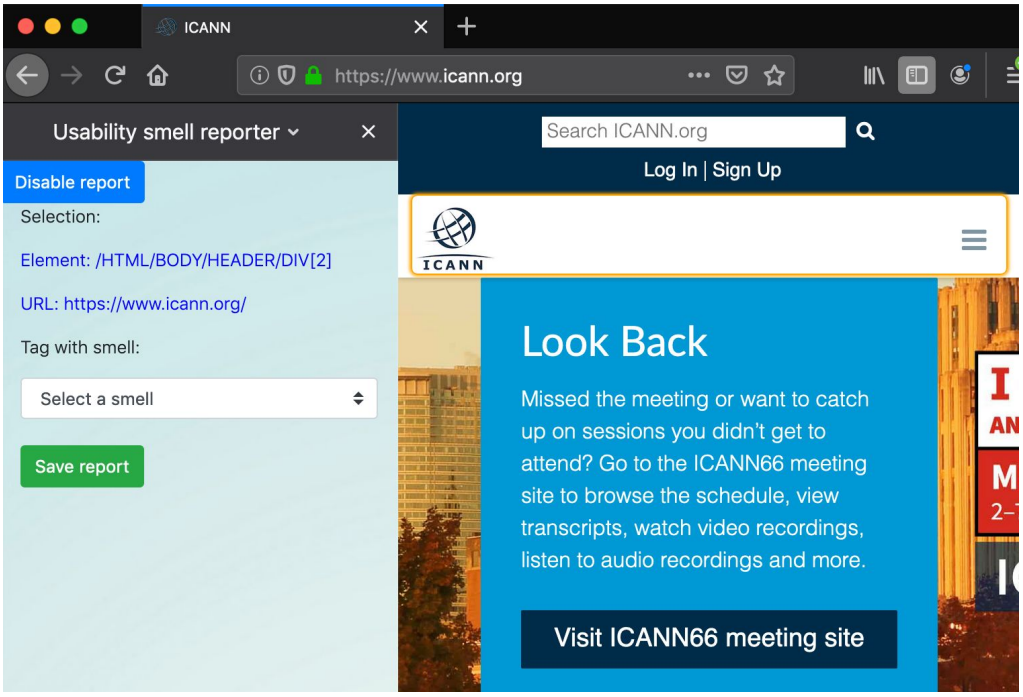


Figura 4.2.3. Demostración de cómo el elemento se pinta de color naranja cuando el mouse pasa por arriba del mismo.

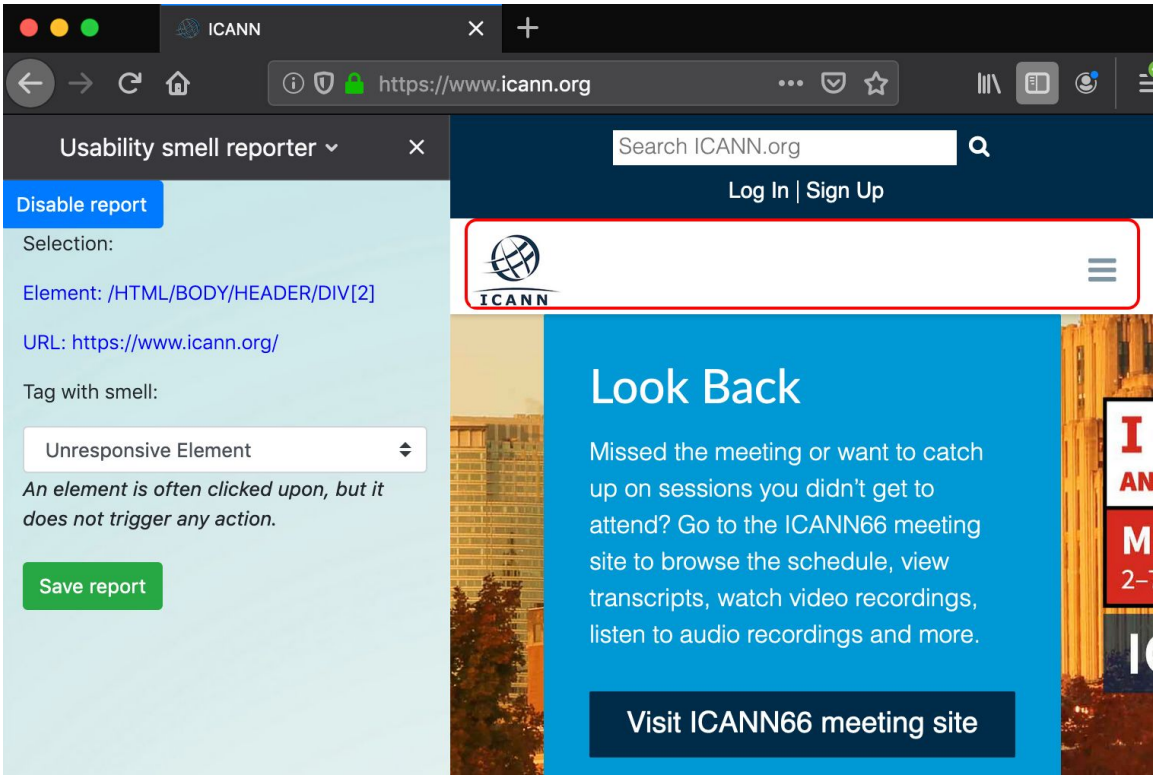
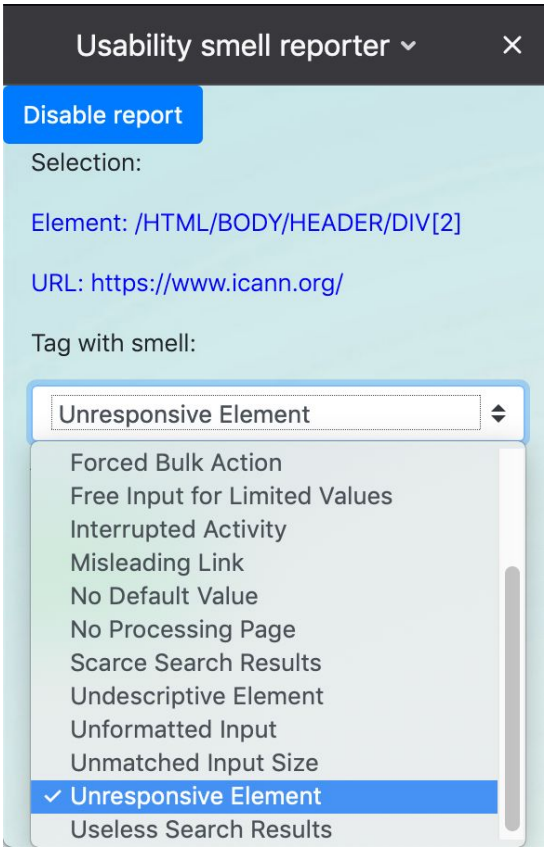


Figura 4.2.4. Elemento a reportar y smell seleccionado.

Una vez seleccionado se verá en la extensión a modo resumen el xPath del elemento y la URL a la cual se asociará el reporte. También se cambiará a color rojo la selección del mismo en la página, tal como se ve en la Figura 4.2.4, para que el voluntario pueda discernir entre una selección y una búsqueda de elemento, en caso de no estar conforme con la selección.

Cuando se habilita el reporte automáticamente se cargarán los smells en la lista (Figura 4.2.5A). Para completar el proceso de reporte el usuario deberá asociar el elemento seleccionado a un *smell*. Cabe destacar que no debe ser técnico informático para poder hacerlo ya que la aplicación ofrece la ayuda necesaria para poder etiquetar el reporte correctamente sin mayor dificultad. Esto permite ampliar la brecha de personas que desean colaborar para mejorar la usabilidad de los sitios webs.

(a)



(b)

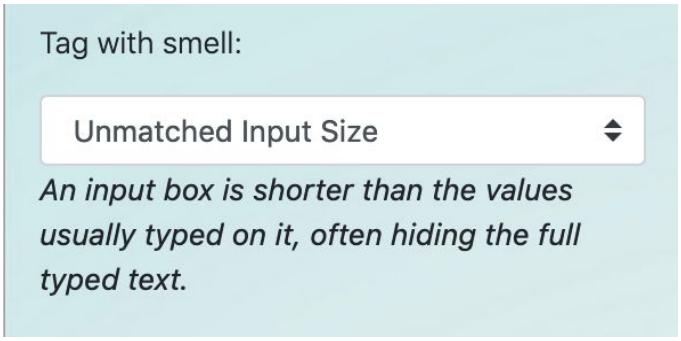


Figura 4.2.5A Listado de usability *smells* para seleccionar. Figura 4.2.5B. Cada usability *smell* posee una descripción que ayudará al voluntario a identificarlo correctamente.

Cuando se selecciona un *usability smell* podrá verse una descripción del mismo que permitirá al voluntario saber si es el correcto para asociar al reporte que está generando en ese instante (Figura 4.2.5B). Dependiendo del tipo de *smell* asociado será el refactoring que se generará al final del proceso para solucionar dicho reporte. Algunos *smells* tendrán asociada una lista de atributos, los cuales son optativos, que en el caso de completarse servirán como complemento para que la generación del refactoring sea más precisa y certera. Estos atributos pueden, o no, ser "repetitivos", es decir, puede existir de 0 a N cantidades. En la aplicación aparecerá la opción de agregar (*add*) otro atributo o eliminar (*remove*) el último que se agregó tal como se ilustra en la Figura 4.2.6(A).

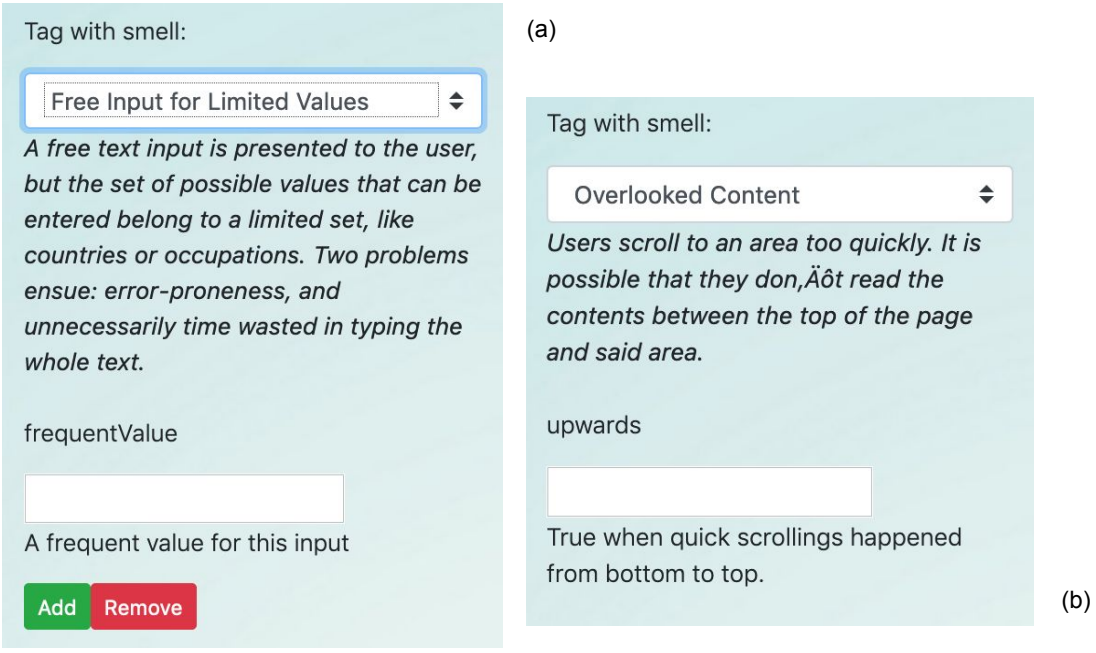


Figura 4.2.6. (A) Smells con atributos dinámicos. Se pueden agregar varias instancias del atributo asociado al smell. (B) Smell con atributo estático. Atributo único asociado al smell y no puede duplicarse.

Otros atributos son estáticos (Figura 4.2.6(B)) y únicos por lo que no se permitirá agregar o quitar más instancias de estos. En el caso de que los atributos se completen se asociarán al reporte y, a la hora de solicitar un refactoring para solucionarlo, se enviarán como complemento del mismo para que la selección de él/los refactoring/s sea más precisa para el problema.

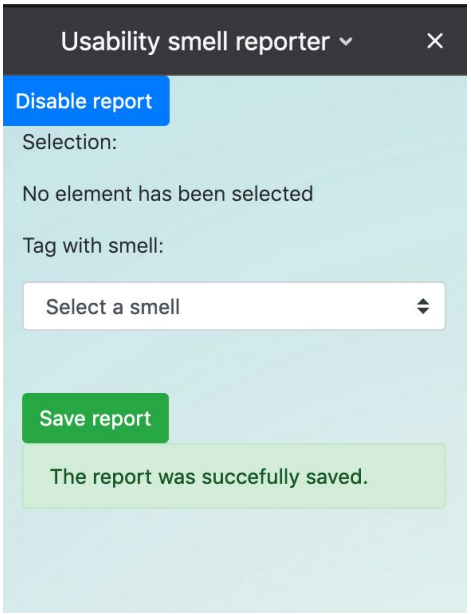


Figura 4.2.7. Guardar reporte. La imagen ilustra un reporte correctamente guardado.

Finalmente presionando el botón *"save report"* se guardará el reporte. Si el reporte se guarda exitosamente, se mostrará un mensaje como el que se ve en la Figura 4.2.7, sino se mostrará un mensaje de error. Todo quedará listo para realizar un nuevo reporte en el caso de ser necesario.

Podría ocurrir que el usuario se encuentre navegando por un sitio sobre el cual sea realizar un reporte y cuando abre la extensión se da cuenta que ya existen refactorings sugeridos como soluciones a reportes realizados (Figura 4.2.8A).

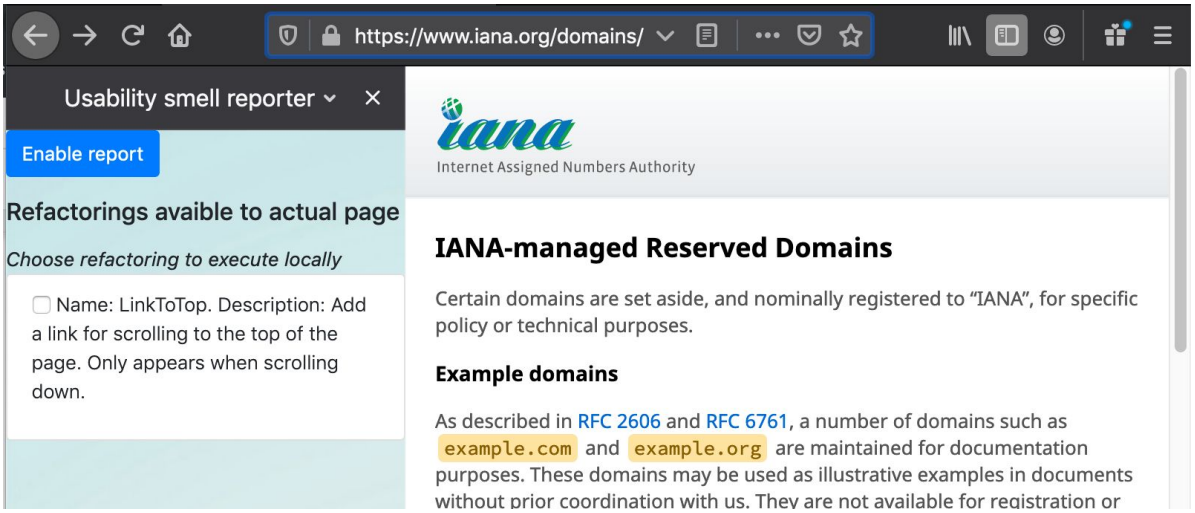
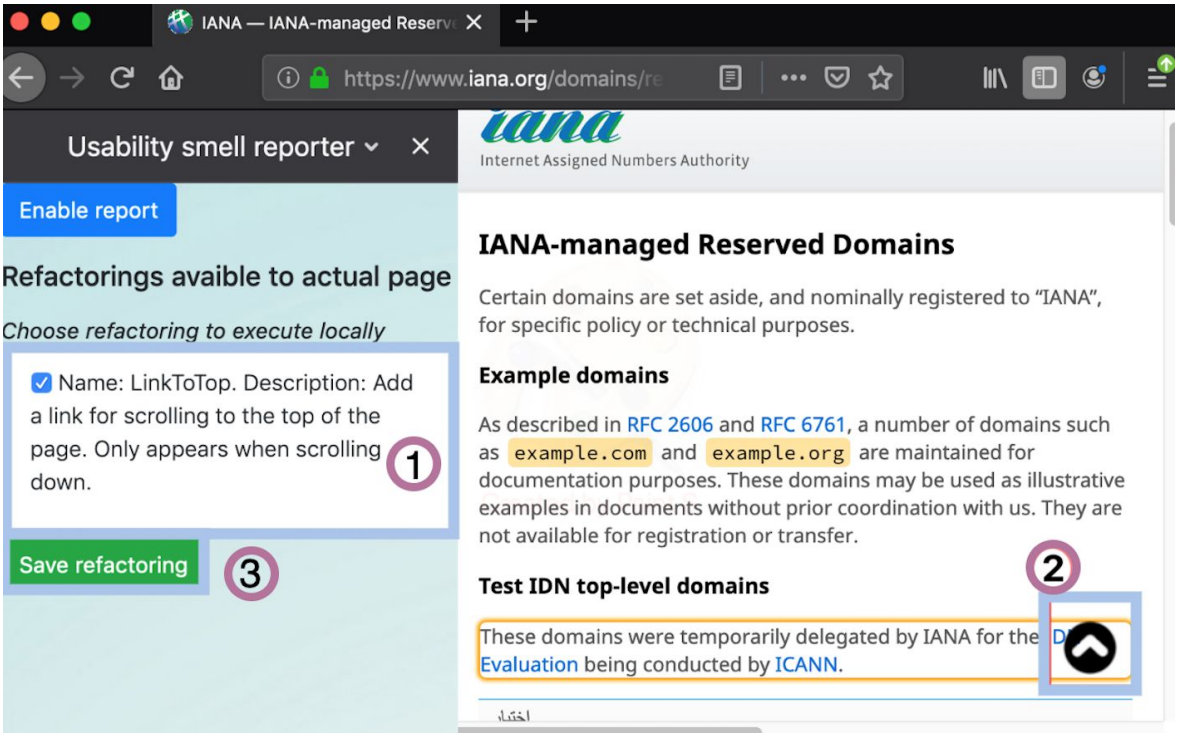


Figura 4.2.8A. Listado de refactorings asociados al dominio actual.



La figura 4.2.8B representa la selección del refactoring ofrecido como solución para esa página (1). Este refactoring se encuentra actualmente ejecutado (2). Si se desea desinstalar el refactoring se debe deseleccionar el *checkbox*. Cabe destacar que la instalación y desinstalación siempre es en el almacenamiento local del navegador del usuario (3).

Presionando la casilla del "*checkbox*" alcanzará para ejecutar el refactoring y guardarlo. Si se desea eliminar el refactoring de nuestra lista local solo bastará con destildar la casilla. Este comportamiento se ve representado en la figura 4.2.8B.

4.3 ¿Cómo funciona?

La aplicación utiliza los eventos *javascript* para realizar las principales tareas. Una vez que el reporte se encuentra habilitado se disparará el evento que se encarga de buscar en la base de datos central si existe algún refactoring asociado a la URL en la que se encuentra el usuario actualmente. Tal como se explicó en la sección anterior, si existen refactorings asociados a la página se listarán y el usuario tendrá la posibilidad de probar las soluciones propuestas y guardarlas localmente.

Para instalar los refactorings de forma local de deberá seleccionar del listado la casilla del "checkbox" y esto disparará el evento javascript necesario para ejecutarlo. Para recuperar el código del refactoring a ejecutar se realiza un requerimiento a la API Kobold con los parámetros necesarios. Una vez recuperado el código se guardará, junto con una serie de metadata asociada al refactoring, en el *local storage*. Si una casilla que se encuentra activa es desactivada entonces se eliminará ese refactoring del almacenamiento local y se refrescará la página para que ya no se vea reflejado ese cambio.

Cada vez que la página se refresca, si hay algún refactoring guardado en el local storage asociado a la URL actual, se ejecutarán y aparecerán tildados automáticamente en el *checkbox*.

La aplicación trabaja en conjunto con la API Kobold. Cuando se habilita el reporte automáticamente se cargarán los *usability smells* en la lista. Para ello se comunicará con Kobold a través de la API ofrecida por éste para obtener la lista completa y actualizada. Uno de los objetivos principales es el de tener centralizada la información sobre los smells y los refactorings asociados a estos y que cualquier cambio sobre estos sea completamente transparente para nuestra aplicación. La comunicación entre las extensiones y Kobold se realizan utilizando el objeto **KoboldClient** que posee funciones específicas para tal fin.

El objeto **StorageManager** es el encargado de gestionar el *local storage* mediante el manejo de la librería *IDBFiles*[7]*.* Cada vez que el usuario selecciona un reporte ya solucionado por otro voluntario, éste se guardará de forma local y para luego ser ejecutado cuando el usuario visite la página. Para que esto sea posible se utiliza el formato JSON para indexar la información necesaria que ayudará a identificar el elemento posteriormente. El formato definido para tal fin es el que se presenta a la figura 4.3.1 a continuación,

[7] IDBFiles: API Javascript que se encarga de realizar la comunicación entre el cliente y el almacenamiento local del navegador para que puedan guardarse datos simples, como json, o complejos, incluyendo archivos.

```
var curFiles = new File([fileContent], fileName, {type: fileMimeType});
var json = {"name" : curFiles.name,
  "kobold_id" : koboldId,
  "description" : "current description",
  "type" : "refactoring",
  "dependencies" : ["AbstractGenericRefactoring"],
  "mime": curFiles.type,
  "exe_priority" : "2",
  "reportId" : reportId,
  "urls" : [ currentUrl ]};
```

Figura 4.3.1. JSON utilizado para indexar el refactoring Javascript con el sitio correspondiente en el local storage.

La variable **curFiles** almacenará el archivo físico con extensión Javascript (.js) que contiene el refactoring en sí. La variable **fileContent** contiene el código javascript recuperado de Kobold utilizando la función *executeRefactoringById* de *KoboldClient*. La variable **json** contendrá toda la información correspondiente a los sitios webs donde deberá ejecutarse el refactoring. Nótese que el atributo **urls** es una lista de direcciones URL ya que el mismo refactoring podría ejecutarse en diferentes sitios webs.

```
async saveFile (storageName, fileName, file) {
    console.log("inicio de saveFile...");
    const storedFiles = await IDBFiles.getFileStorage({name: storageName});
    await storedFiles.put(fileName, file);
    console.log("fin de saveFile...");
  }
```

Figura 4.3.2 Función que se encarga de guardar el refactoring en el *local storage* utilizando la API *IDBFiles*.

Una vez completados los datos necesarios se procede a guardar el *script* y su metadata en el almacenamiento local. Para ello se utiliza la función presentada en la figura 4.3.2 consumiendo la librería IDBFiles.

También existe un manejador de scripts, **Execute*ScriptManager***, encargado de gestionar todos los *scripts* ejecutados desde la extensión hacia el sitio web. Mantiene el status de los refactorings ejecutados actualmente. Cuando un refactoring se ejecuta y se guarda en el local storage, se agregará a la lista y cuando se elimina del *local storage*, se eliminará de la misma. La ejecución del código javascript desde la extensión hacia la página web es posible gracias a la función `browser.tabs.executeScript` provista por la API WebExtension. Esta función se encarga de ejecutar código Javascript desde la extensión del navegador (*Usability Smell Reporter*) hacia los tabs abiertos de navegador, dependiendo el tipo de configuración que se haya realizado permitiendo la comunicación desde la extensión hacia la página web en cuestión.

El manejador central, el **ManagerScript**, es el encargado de gestionar los manejadores mencionados anteriormente para lograr que todo el flujo se ejecute correctamente. Una vez que toda la información es recopilada el ManagerScript se encargará de guardar el refactoring en el almacenamiento local utilizando la función `saveFile` y luego ejecutará el refactoring utilizando la función `executeScript`.

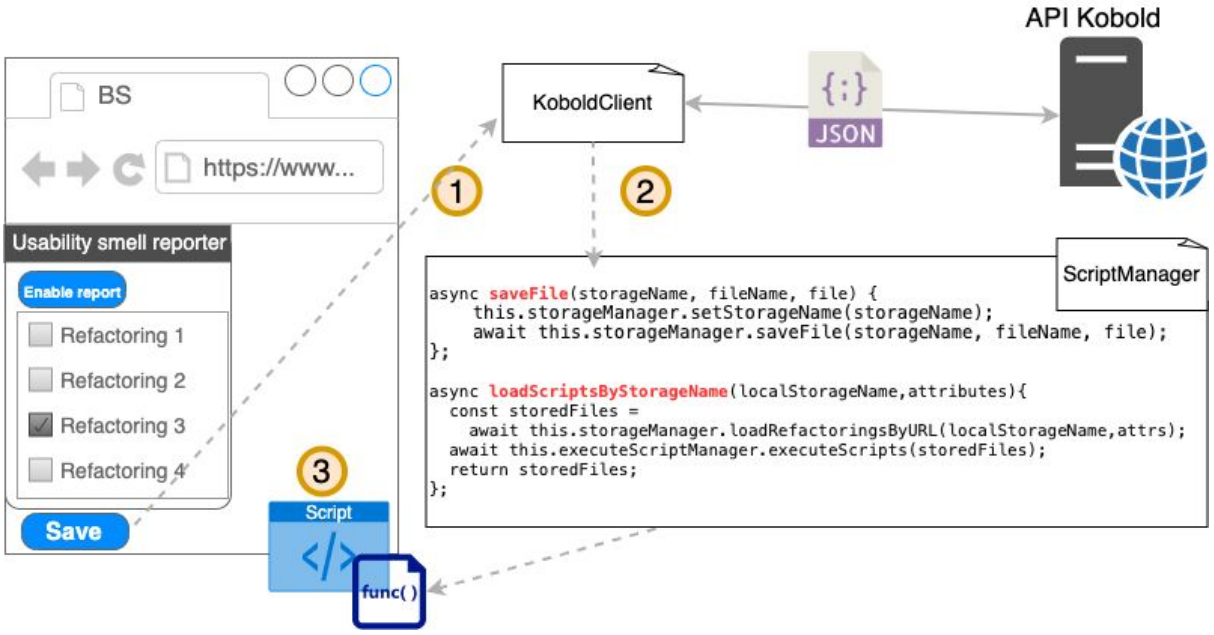


Figura 4.3.3. Flujo completo de ejecución y guardado del refactoring.

En la figura 4.3.3 se puede observar el flujo de ejecución y guardado de un refactoring seleccionado. Cuando el usuario decide guardar el refactoring se dispara un evento para pedirle a Kobold el código Javascript del refactoring elegido. Luego éste se guardará en el almacenamiento local para luego ser ejecutado del lado del cliente.

El reporte final se guardará en la base de datos SQLite3 donde se centralizarán todos los reportes realizados. Esto permitirá que puedan gestionarse independientemente de dónde se generen los reportes o quiénes los hagan.

Para guardar los reportes se utiliza la API Python desarrollada para interactuar entre las extensiones y la aplicación web de gestión de reportes la cual veremos en el siguiente capítulo. En el reporte generado se guardará la url afectada junto con el xPath para poder asociarlo luego y aplicar el refactoring para su solución. Adicionalmente se asociará el *smell* y los atributos en el caso en que el usuario los hubiera completado.

CAPÍTULO 5

Usability Smell Manager

5.1 Introducción

Cuando se guarda un reporte éste podrá verse reflejado en la aplicación web *Usability smell manager* para su eventual gestión. Esta aplicación ofrece una API para poder comunicarse con las extensiones con las cuales interactúa.

La aplicación web está desarrollada con el framework Django que utiliza el lenguaje de programación Python y toda la información que necesita ser guardada se almacenará en SQlite3, sistema de gestión de base de datos.

Los voluntarios que deseen colaborar con la resolución de los reportes realizados deberán comenzar desde este sitio web y adicionalmente descargar la extensión con la que trabaja en conjunto: *"Usability smell fixer"*. Ambas aplicaciones son libres y gratuitas. No se necesita usuario registrado para acceder a estas.

Usability smell manager es fácil y sencilla de utilizar y, tal como las otras aplicaciones, no se necesita conocimiento técnico para poder usarla solo basta con ingresar al link correspondiente para comenzar. Esto es una gran ventaja para que el voluntario se vea incentivado para continuar con el flujo de reporte y solución del mismo para mejorar la usabilidad del sitio web que le interesa.

5.2 ¿Cómo se utiliza?

El usuario voluntario deberá ingresar al sitio web para ver el listado de reportes realizados mediante la url http://localhost:8001/smells (figura 5.2.1). En dicho listado se podrá observar la URL de la aplicación web donde se encuentra el problema de usabilidad reportado, el nombre del *smell* asociado a este y la fecha en la cual fue realizado el reporte. El voluntario podrá identificar fácilmente cada registro lo cual le permitirá elegir con libertad qué reporte desea solucionar, podrá elegir el que sea de su interés y de esta forma ayudar a mejorar la usabilidad del sitio sobre el cual se hizo el reporte.

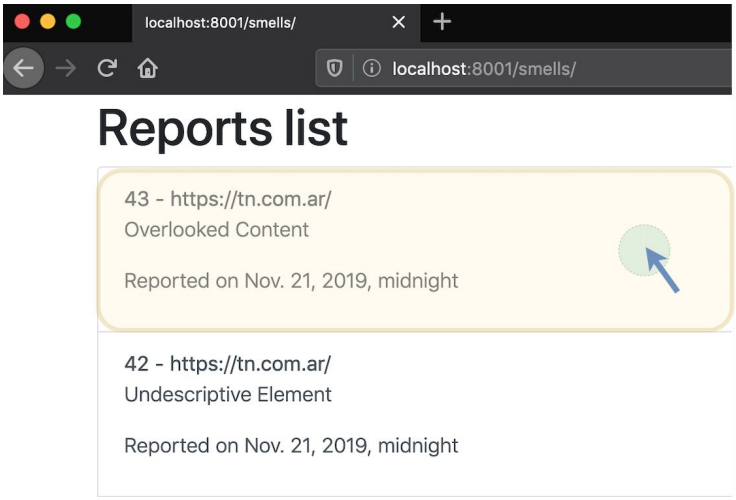


Figura 5.2.1. Imagen ejemplo del listado de reportes en *Usability smell manager.*

Una vez que seleccione la opción deseada podrá utilizar la aplicación *Usability Smell Manager* para solucionar el problema.

Cabe destacar que no existe restricciones para los usuarios, es decir, cualquier usuario podría ingresar al sitio y realizar la mejora. Un voluntario podría utilizar la aplicación *Usability smell reporter* y luego ingresar a *Usability smell manager* para solucionar el reporte que él mismo realizó. Esto permite que el flujo de reporte y resolución puedan ser más rápidos y no tengan que esperar a que otro usuario ingrese al manager y solucione el problema.

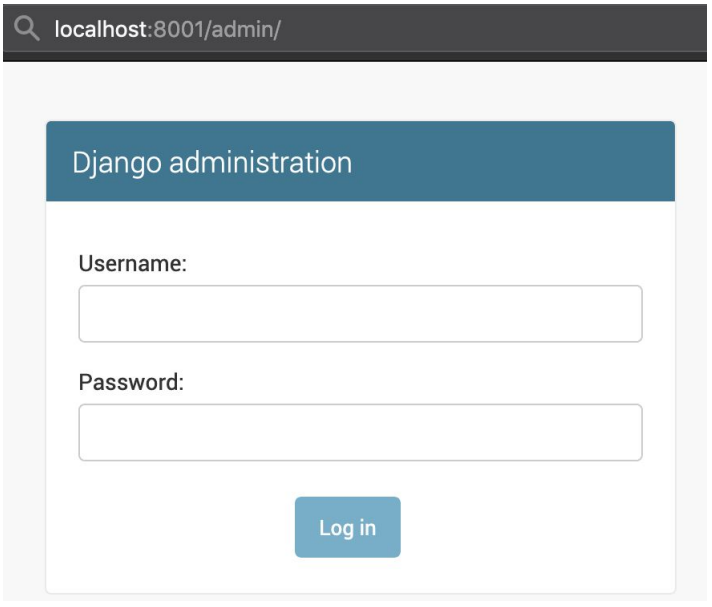


Figura 5.2.2. Interfaz de usuario para login admin.

Usability smell manager también ofrece una interfaz de usuario para realizar la administración de la base de datos. Ingresando a la dirección http://localhost:8001/admin/ se podrá acceder a ella con las credenciales correspondientes tal como se muestra en la figura

5.2.2. En esta base de datos se encuentra toda la información relativa al flujo de reportes y resoluciones de *usability smells*. Se mantendrá una bitácora de reportes y a través de estos se podrá identificar aquellos que fueron solucionados y de esta forma solicitar el refactoring para ejecutar utilizando los valores asociados correspondientes al mismo.

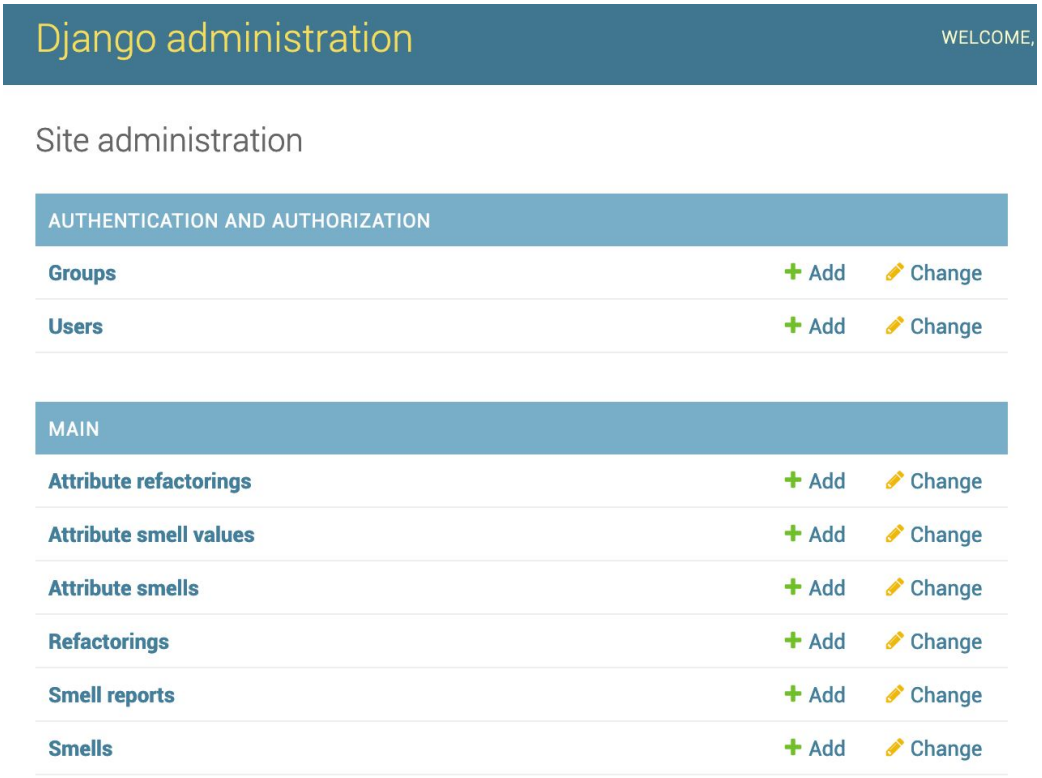


Figura 5.2.3. Interfaz de usuario para administración de la base de datos.

La figura 5.2.3 ilustra la página principal del administrador. Desde allí se podrá gestionar tanto las tablas asociadas a los reportes como las tablas asociadas a los usuarios del sistema. En nuestro caso solo se manejan usuarios para el ingreso al administrador de base de datos, no al sitio web que gestiona el listado de reportes.

5.3 ¿Cómo funciona?

Usability smell manager trabaja en conjunto con la extensión *Usability smell fixer* a través de eventos javascript configurados para un determinado propósito.

```
<script type="text/javascript">

function onSolve(report, href){
  var sendDataCrossSite = document.querySelector('#sendedDataCrossSite');
  sendDataCrossSite.value=report;
  sendDataCrossSite.click();
  window.location = href;
}
</script>
```

Figura 5.3.1. Extracto de código de *Usability smell manager*. Función a través de la cual se comunica la aplicación web y la extensión *Usability smell fixer.*

La comunicación entre ambas aplicaciones se realiza mediante la configuración de un *content script* utilizando las herramientas que provee la *API Javascript WebExtension* para dicho propósito. En este archivo se agrega código *Javascript* que se encuentra constantemente escuchando los eventos realizados en el sitio en cuestión. En nuestro caso, el *content script* estará atento al evento "click" de la función `onSolve` (figura 5.3.1) que se encuentra en la página principal del listado de reportes en la aplicación *Usability smell manager*. Cuando se hace *click* sobre un reporte se enviará a la extensión toda la información necesaria sobre este para que pueda ser solucionado. Cómo funciona esta comunicación la veremos en detalle en el próximo capítulo.

Para el desarrollo de este gestor se utilizó el servicio que ofrece el framework Django [Django05]. Para esto se realizó de la configuración del modelo y del archivo admin.py, donde se configuran las clases que se desean agregar al administrador. No es necesario exponer el modelo completo, puede configurarse y personalizarse para sólo mostrar el contenido deseado y de esa forma limitar los valores que son posibles gestionar y a su vez ocultar aquellos que no desean que sean públicos o accesibles de ningún modo.

CAPÍTULO 6

Usability smell fixer

6.1 Introducción

Herramienta colaborativa desarrollada con *API Javascript WebExtension*. Es una extensión libre y gratuita y puede ser utilizada por cualquier usuario que desee colaborar en la resolución de los smell reportados para un determinado sitio web a través de la herramienta *Usability smell reporter*.

Usability Smell Fixer trabaja en conjunto con el sitio web *Usability Smell Manager*. Ambas aplicaciones se comunican entre sí para compartir la información sobre el reporte que el voluntario desea solucionar.

Esta herramienta es el último paso dentro del flujo de reporte y solución de los malos olores de usabilidad que un voluntario puede identificar en un determinado sitio web.

Una vez que se decide cuál es el refactoring adecuado para el problema reportado se guarda la solución y se actualiza el reporte con el refactoring elegido. Esto permitirá que dicha solución se vea reflejada como nueva opción en la extensión *Usability smell reporter* para ser seleccionada en caso de que el usuario desee instalarla en su navegador de forma local.

Una de las grandes ventajas que ofrece la aplicación es la de permitirle al usuario probar los refactorings antes de decidirse por uno como solución. Esto se debe a que para un determinado *usability smell* pueden existir más de un *usability refactoring.* El voluntario podrá seleccionar de a uno y deberá cargar los atributos que éstos requieren, en caso de corresponder, para luego ejecutarlo y decidir si es la solución que más se adapta a sus necesidades.

En este trabajo se desarrollaron los refactorings Split Page y Distribute Menu como parte de la contribución a la comunidad. Estos refactorings fueron entregados a Kobold para que de esta forma puedan ser obtenidos cuando se solicita el listado de refactorings asociado a un determinado *usability smell*.

6.2 ¿Cómo se utiliza?

Una vez que el usuario selecciona del listado del manager el reporte que le interesa resolver, deberá abrir la extensión y de esta manera verá allí la información del reporte correspondiente junto a un listado de refactorings asociados al smell.

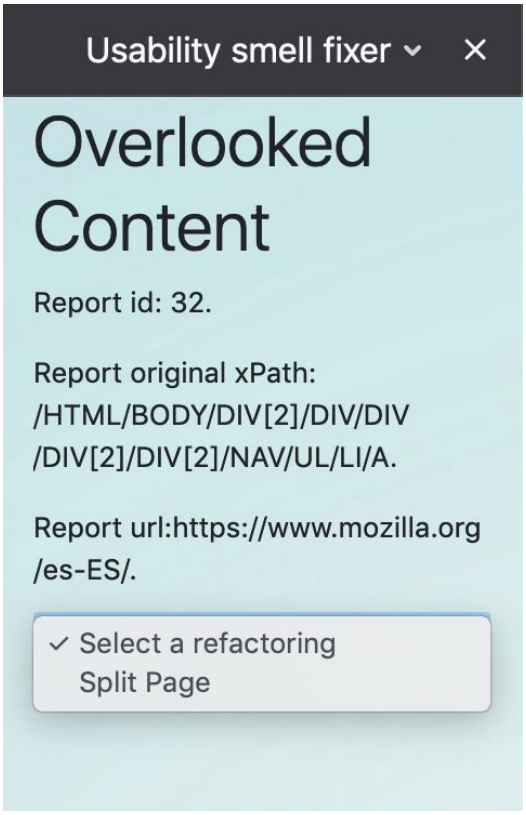


Figura 6.2.1.Pantalla inicial Usability Smell Fixer.

Como muestra la figura 6.2.1 se verá detallada la información del reporte, el nombre del *smell*, la URL correspondiente y el listado de refactorings disponibles para el *smell* reportado.

Al seleccionar un refactoring, en caso de corresponder, se desplegará un listado de parámetros necesarios para poder ejecutar el código y probar temporalmente cómo se vería reflejado en la página. Los parámetros son obligatorios y, en algunos casos, repetibles, es decir, el voluntario podría enviar más de una instancia del mismo atributo con valores diferentes. A continuación se presenta el ejemplo del caso del *Split Page*.

Como se muestra en la figura 6.2.2, el refactoring *Split Page* necesita como mínimo dos atributos, la sección principal y otra sección general. En el caso de que el voluntario desee agregar más secciones deberá presionar el botón “add” y cargar los valores respectivos. Si el voluntario desea eliminar alguna sección extra agregada, presionando el botón “remove" será suficiente para eliminar la última sección agregada. Cabe destacar que el voluntario puede eliminar todas las secciones que haya agregado, excepto las dos que son obligatorias.

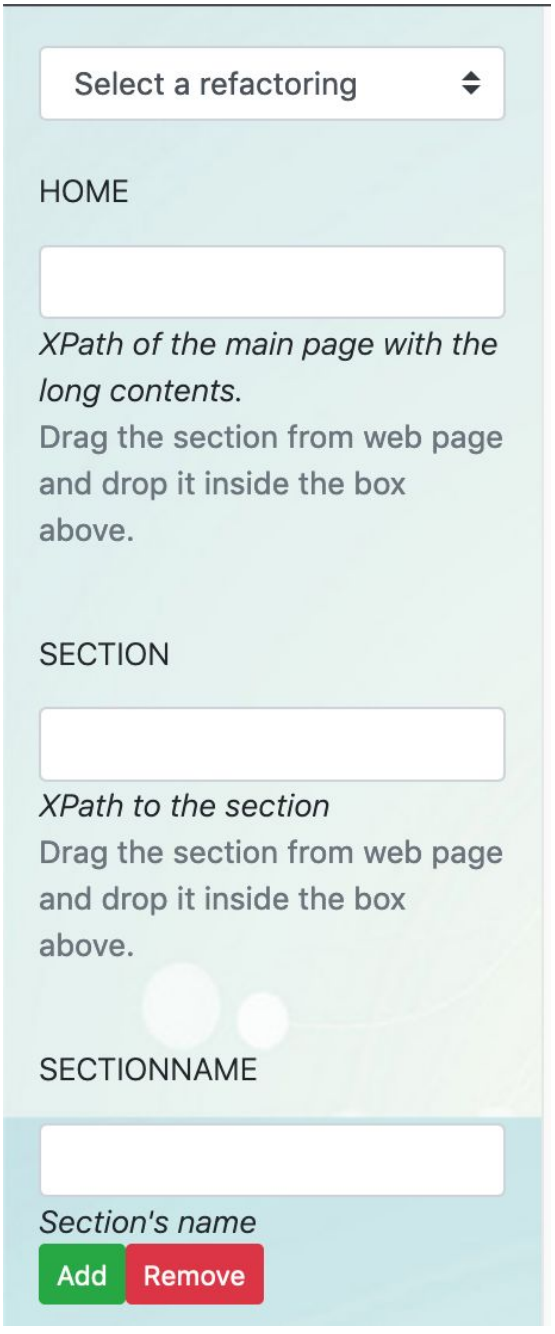


Figura 6.2.2. Ejemplo de carga atributos para *Split Page*.

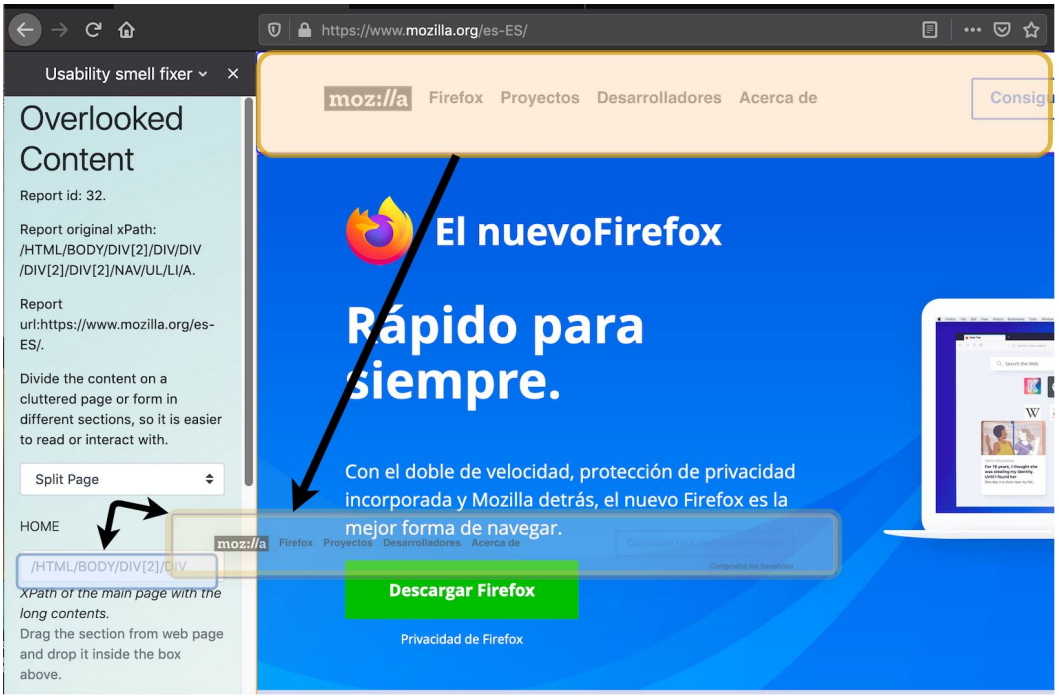


Figura 6.2.3. Drag and drop para la selección de secciones.

A la hora de elegir una sección, se podrá utilizar la funcionalidad drag and drop que consiste en arrastrar el elemento deseado al campo de texto correspondiente al xPath de la sección. Tal como se ve en la figura 6.2.3 solo es necesario arrastrar y soltar el elemento dentro del campo de texto para que éste se autocomplete con el valor de xPath. Esto permite mayor comodidad al voluntario a la hora de seleccionar el elemento deseado y no hace falta grandes conocimientos técnicos para cargar el valor del xPath, provocando más satisfacción a la hora de utilizar el sistema.

Una vez que se cargaron todos los valores necesarios de los atributos del refactoring, el voluntario tendrá la posibilidad de probarlo para ver cómo quedará ejecutado en la página web. Para ello solo bastará con presionar el botón “refactorizar” para ver los resultados en el sitio. Esta herramienta es muy potente, ya que le permite al usuario probar los refactorings antes de considerar solucionado el reporte. En caso de no estar contento con los resultados, tiene la posibilidad de cambiar los valores cargados para el refactoring que está probando o bien elegir otro refactoring de la lista para probar opciones diferentes.

A continuación presentaremos como ejemplo al refactoring Split Page. Se podrá ver el sitio en su estado normal y luego de cargar las secciones correspondientes se verá el resultado final de la transformación del sitio web.

La figura 6.2.4 presenta el sitio web reportado por el voluntario con el *bad smell Overlook content*. Este *smell* sugiere como refactoring para su solución al *Split Page*. Una vez seleccionado dicho refactoring se deberá cargar las secciones necesarias, en nuestro ejemplo se cargarán las dos secciones obligatorias.

Figura 6.2.4. Selección de secciones para el refactoring.

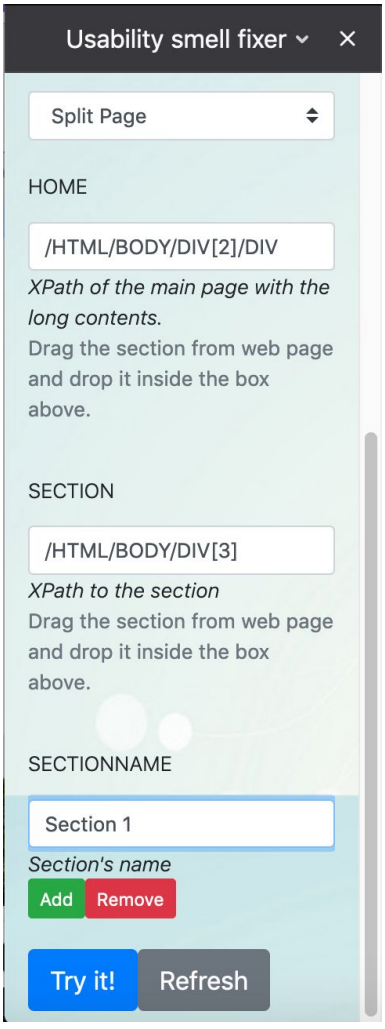


Figura 6.2.5. Selección del refactoring Split Page y carga de valores para cada sección utilizando drag and drop como estrategia de selección para las distintas secciones.

En la figura 6.2.5 se muestran los datos cargados de ambas secciones.

Cuando el usuario finalmente selecciona todos los valores y presiona el botón “Try it!” podrá ver en la página web el refactoring ejecutado. Una vez ejecutado el refactoring lo primero que se verá será el contenido de la sección principal y el listado de secciones en el margen superior izquierdo.

Figura 6.2.6. Refactoring Split Page ejecutado. Sección principal. Resaltado en naranja la sección principal. El resto del contenido desaparece y solo queda esta sección a la vista.

Todo el contenido original del sitio web se ocultará y se desplegará automáticamente solo el contenido de la sección principal tal como se ve en la figura 6.2.6. Luego si presiona la siguiente sección "Section 1" se ocultará la sección principal y se desplegará la sección 1 (Figura 6.2.7).

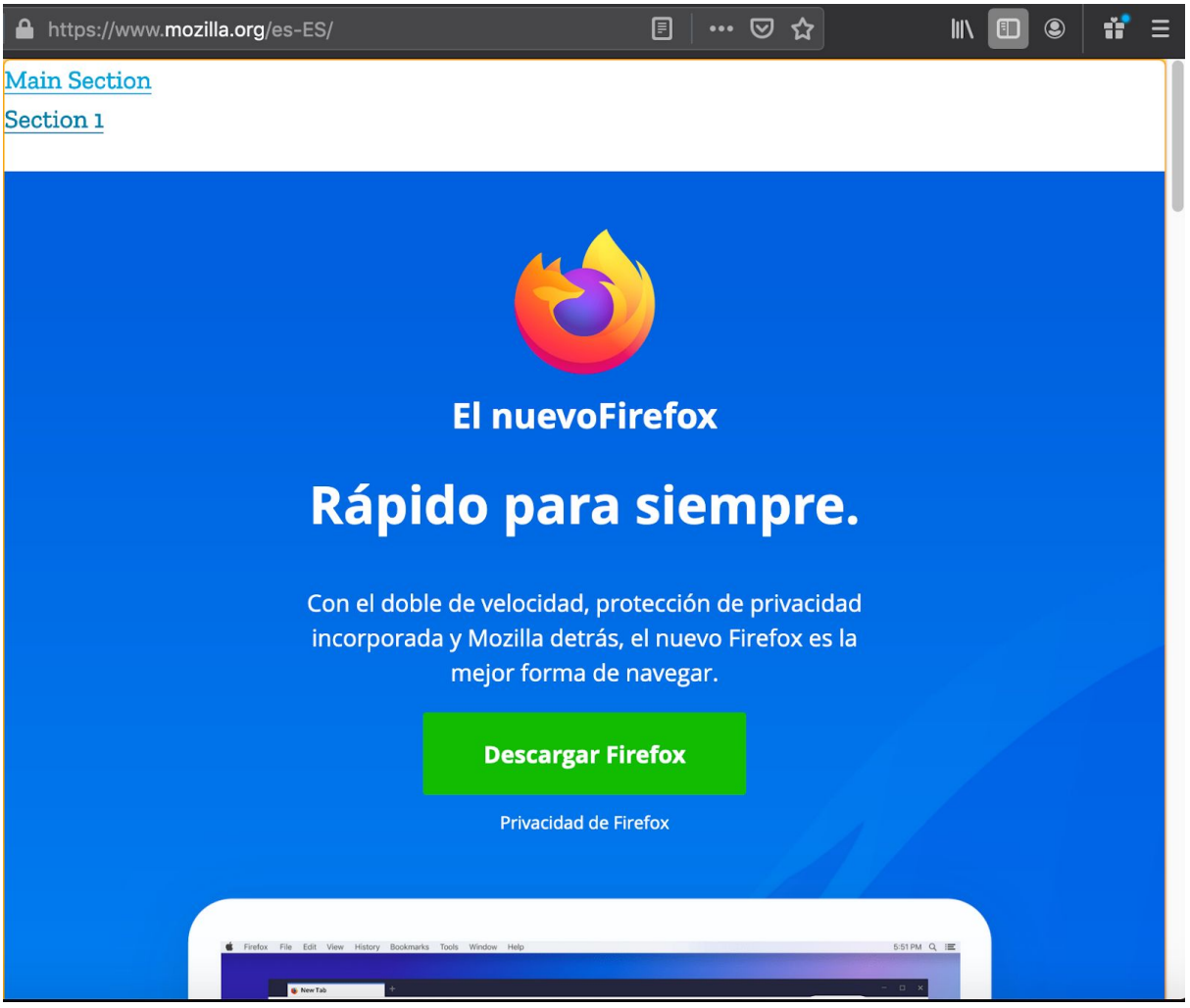


Figura 6.2.7. Refactoring Split Page ejecutado. Sección 1 seleccionada.

El usuario podrá navegar por las diferentes secciones. Cada vez que cambie de sección se ocultará la anterior y se desplegará la nueva. En cada una de ellas el menú estará visible para poder acceder a cualquier sección. La gran ventaja de este refactoring es que le permite al usuario quitar contenido excesivo que probablemente la mayor parte del tiempo no utiliza o provoca que se pierda y no encuentre lo que desea. Esto también trae acarreado problemas para las personas que utilizan screen reader para leer la página web. Tanto contenido puede provocar que tarden demasiado tiempo en hacer una acción simple o quizás al utilizar funcionalidades de diseño que ocultan contenido también puede traer problemas y quizás nunca lleguen a acceder a este.

Finalmente cuando el voluntario decide que ha encontrado el refactoring indicado para solucionar el reporte, presiona el botón “save refactoring”.

De esta forma concluye el flujo de reporte y solución de un *smell* en un determinado sitio web. A partir del momento en que se guarda el refactoring, éste se asociará al reporte y estará disponible para ser utilizado desde la aplicación *Usability Smell Reporter.*

6.3 ¿Cómo funciona?

La comunicación entre la aplicación Usability Smell Manager y Usability Smell Fixer sucede gracias al *content script* y el *background script* que ofrece la extensión de firefox *API WebExtension Javascript*, los cuales se encargan de intercambiar información entre la página web y la extensión.

```
/** Esta funcion carga el evento para escuchar al input que contiene
 la informacion del reportedSmell a solucionar desde la app SmellFixer */
function loadSendedDataCrossSite(){
  var sendedData = document.querySelector('#sendedDataCrossSite');
  if (sendedData != undefined){
    sendedData.addEventListener('click', function(event){sendedDataMsg(event, this)});
  }
};

function sendedDataMsg(evento, self){
  browser.runtime.sendMessage({ idReport  : self.value});
}
```

Figura 6.3.1. Extracto de código *Usability smell fixer.* Función que recepciona la información enviada desde la aplicación web *Usability smell manager.*

El extracto de código presentado en la figura 6.3.1 corresponde a un archivo *content script* en la extensión *Usability smell fixer* desarrollada con la API WebExtension Javascript, éste se instala del lado del cliente y permite la comunicación entre ambas partes, el cliente web y la extensión de navegador. Debido a que se trata de un archivo javascripts en el lado del cliente, el sitio web tendrá acceso a este y viceversa y es por eso que al agregar el listener al selector con identificador `sendDataCrossSite`, cuyo elemento original se encuentra en el sitio web *Usability web Manager*, se podrá escuchar la acción *click* sobre este. Esto se realiza para que, una vez disparada la acción, se acceda a la función encargada de escuchar el evento la cual se encarga de enviar el mensaje al *background script* a través de la función `browser.runtime.sendMessage({idReport : self.value });`. El valor que envía corresponde al identificador del reporte realizado. La función `browser.runtime.sendMessage` la provee la API WebExtension para poder enviar información desde un *content script* hasta un *background script*, es decir, desde un sitio web hacia la extensión del navegador.

```
browser.runtime.onMessage.addListener(function(report) {
  if(report.url == undefined) {
    // Load the refactorings from Kobold by ID to fix the report
    loadRefactoringsFromKobold(report);
  } else {
    onClickClientEvent(report);
  }
});
```

Figura 6.3.2. Extracto de código *Usability smell fixer.* Evento background script configurado para escuchar el evento content script proveniente de la aplicación *Usability smell manager.*

Desde el background script se recepcionará la información mediante la función `browser.runtime.onMessage.addListener`. Esta función se encarga de escuchar a todos los *content script* configurados y cuando alguno le envía información a través de la función `sendMessage`, recepcionará todos los mensajes en el mismo listener, es por ello que se debe tener especial cuidado en la programación de éste ya que si se envían varios mensajes desde varios *content script* se debe realizar un trato especial para saber desde cuál estos viene. En nuestro caso, como se ve en la figura 6.3.2, sólo recibimos información de un único content script.

Cuando se realiza la carga inicial de la aplicación se utilizarán los valores enviados por Usability Smell Manager para rellenar los campos necesarios. El listado de refactorings se pedirá a Kobold a través de su API. Para obtenerlo se enviará el identificador del smell correspondiente en Kobold y, en el caso que el usuario haya cargado valores, el listado de atributos correspondientes. Estos atributos ayudan a Kobold a realizar filtros más precisos y poder devolver una lista reducida de refactorings que ayudarán a solucionar el problema con mayor exactitud. En caso de no cargar valores en los atributos del smell, Kobold devolverá el listado completo de refactorings asociados al *smell*.

Existen tres tipos de refactorings a tener en cuenta a la hora de recepcionar la lista. Aquellos refactorings que no necesitan ningún atributo obligatorio. Los refactorings que solicitan atributos obligatorios y aquellos refactorings que poseen atributos obligatorios pero a la vez dinámicos, es decir, pueden enviarse de 1 a N instancias del mismo.

Abordaremos el caso de los atributos dinámicos. La respuesta de Kobold para estos atributos corresponderá a un arreglo de objetos JSON, es decir, aquella respuesta que contenga un arreglo de objetos se presentará al usuario como un atributo dinámico donde se le ofrecerá agregar una o varias instancias del mismo conjunto de objetos. Se explicará mejor con la figura 6.3.3 a continuación.

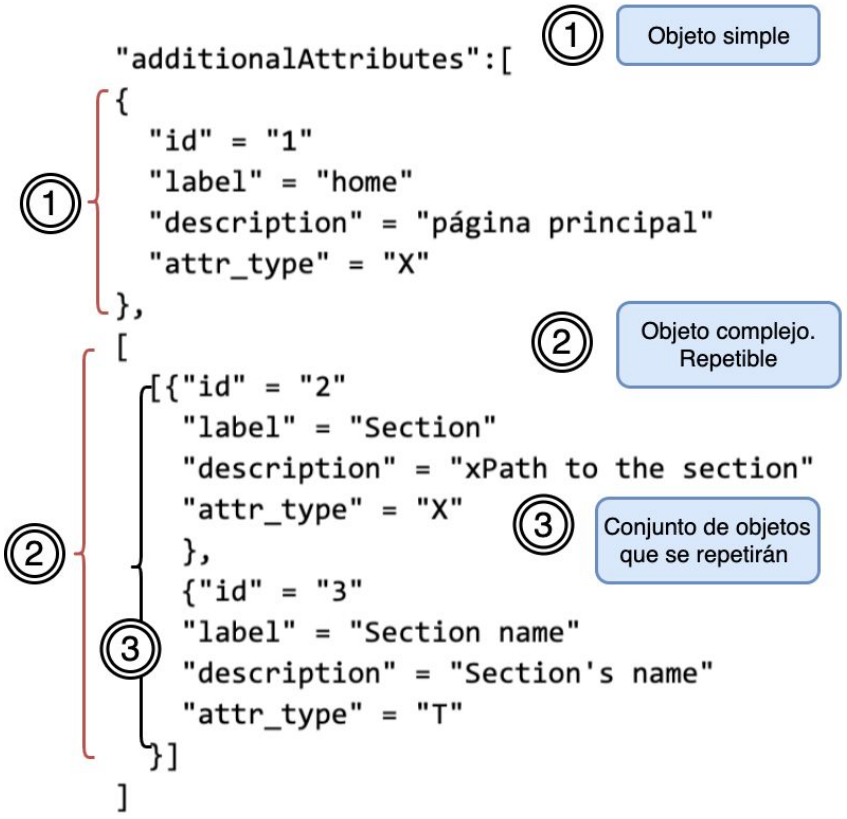


Figura 6.3.3. Atributos adicionales devueltos por Kobold dinámicos.

El punto 1 muestra el objeto estático en este caso será un campo de texto de tipo xPath con *label* "home". Este atributo aparecerá solo una vez y es obligatorio. El punto 2 nos muestra un arreglo que posee dentro los arreglos que serán repetibles. Es decir, todos los arreglos que se encuentren dentro del punto 2 serán obligatorios al menos una vez y podrán agregarse N veces. El punto 3 muestra el arreglo que contiene el primer, y único en nuestro ejemplo, conjunto de objetos que conforman el atributo repetible. Se dibujará en la pantalla dos campos de texto uno de tipo xPath con la leyenda "xPath to the section" y otro de tipo texto con la leyenda "Section's name". Ambos deben repetirse conjuntamente.

Cuando se presiona el botón de guardado automáticamente se guardará la resolución en la base de datos SQLite3. Cabe destacar que es la misma base de datos que se utiliza para el flujo completo de reporte y resolución de smells. Esto permite que la información se encuentre centralizada y pueda ser gestionada sin problema. *Usability smell Fixer* utilizará la API de *Usability Smell Manager* para guardar los valores del refactoring y asociarlo al reporte. Del refactoring solo se guardarán los atributos y sus valores y el id correspondiente a éste en la base de datos de Kobold. Esto se debe a que, desde la aplicación *Usability Smell Reporter* se utilizarán estos valores para pedirle a Kobold el código javascript del refactoring a ejecutar en el caso de que el voluntario desee utilizar de manera local el refactoring.

Ambas extensiones se comunican constantemente con la API Kobold para mantener la transparencia en el caso de que algún refactoring o algo smell cambie. De esta manera las extensiones no sabrán del cambio y tanto los smells como los refactorings se encontrarán siempre actualizados y no será necesario mantener los códigos por separado.

Split Page

Existen sitios webs que contienen demasiada información en una misma página, esto se llama saturación de contenido lo cual provoca confusión en la navegación para el usuario. Para resolver este problema se define el refactoring *Split Page* [Garrido13a], el cual divide el contenido de la página en pequeñas secciones o en diferentes páginas y de esta forma facilita el acceso a la información.

Para aplicar este refactoring son necesarios dos parámetros:

- URL en donde se aplicará el refactoring.
- Listado de secciones que se deseen crear. En el listado debe especificarse en concepto de xPath la posición del elemento que se desea dividir y el nombre que se le quiere asignar a cada sección. Cada *Split Page* tendrá una sección principal, la cual podrá ser accedida desde todas las secciones permitiendo de esta forma volver al inicio de las secciones si se desea cambiar a otra.

El refactoring fue desarrollado y provisto a Kobold para que se agregue al listado de refactorings que gestiona. A continuación se presenta la respuesta de la API Kobold cuando sugiere el refactoring Split Page como solución al BS *Overlook Content y Abandoned Form*.

```
[{
   "id" : 4,
   "name" : Split Page,
   "description" : Divide the content on a cluttered page or form in different
   sections, so   it is easier to read or interact with.,
   "additionalAttributes" : [{
          "description" : XPath of the main page with the long contents.,
          "id" : 1,
          "type" : X,
          "label" : home
       },[[
        {
          "description" : XPath to the section,
          "id" : 2,
          "type" : X,
          "label" : section
        }, {
          "description" : Section's name,
          "id" : 3,
          "type" : T,
          "label" : sectionName
```

```
        }
      ]]
    ]
}]
```

Una limitación para estas soluciones es que el xPath utilizado para identificar el elemento afectado debe ser absoluto y no debe cambiar ya que estos identifican a los elementos involucrados en el refactoring, es por ello que se debe tener especial cuidado cuando se instancian varios refactorings para un mismo sitio.

El refactoring Split Page se presenta como solución a los smells overlook content y abandoned form.

- *Overlook Content* solución aplicando refactoring *Split Page*: el usuario voluntario solo deberá seleccionar las secciones que desea crear y su contenido en concepto de xPath, sin necesidad de tener conocimientos sobre el lenguaje de programación que existe por detrás de la creación de este refactoring. La selección de cada sección se podrá hacer mediante la función *drag and drop* para que sea más fácil e interactivo para el usuario. Una vez definida la solución, esta misma podrá ser utilizada por todos los usuarios de la extensión "*Usability Smell Reporter*" para ejecutarla de forma local en su navegador.

- *Abandoned form* solución aplicando refactoring *Split Page:* una solución aplicando Split page, es dividir el formulario en varias páginas para que el usuario vaya completándolo de a poco. Pero esto es demasiado complejo y costoso ya que se ve involucrada la interacción con el servidor.

El ejemplo de este refactoring aplicado se mostró en la sección 6.2.

Distribute Menu

Otro de los refactorings desarrollados en este trabajo es el *Distribute Menu* [Garrido13a]. Se enfoca en distribuir la funcionalidad aplicada a varios elementos por una acción global, dicho de otra manera, una acción que afecta a varios elementos. Esto puede ocasionar problemas en la accesibilidad del usuario, debido a la necesidad de seleccionar los elementos para luego aplicar la acción. Un ejemplo sería con los usuarios que utilizan un *screen reader*, las acciones que deben realizar para poder seleccionar la lista de elementos que desean modificar son muy tediosas para finalmente lograr el objetivo.

Este refactoring apunta a aplicar estas acciones globales a cada elemento de forma individual de manera que el usuario no necesite volver hasta el inicio para ejecutarlas ya que las tendrá inmediatamente a mano luego de leer el elemento.

Una alternativa a este refactoring es *Postpone Selection* que posiciona la selección de cada elemento (por ejemplo un checkbox) al final del mismo. Esto se debe a que, cuando se utiliza un screen reader se lee primero la ocurrencia del *checkbox* y luego el contenido del registro, por lo cual, si el usuario desea seleccionarlo deberá volver para atrás. Con este refactoring eso no será necesario ya que se leerá primero el contenido y luego figurará el *checkbox* para poder seleccionarlo.

El refactoring *distribute menu* se presenta como solución al usability *smell forced bulk action*.

- *Forced bulk action* (acción en lote forzada) solución aplicando *Distribute menu*: se deberá especificar la URL correspondiente al sitio donde se aplicará el refactoring luego especificar. Se debe especificar el listado donde se encuentran los elementos en donde se distribuirá el menú de acciones. Finalmente se debe indicar cuál será la acción a distribuir, este se colocará al final de cada elemento de la lista de modo que puedan realizar las acciones de forma individual.

El refactoring fue desarrollado y provisto a Kobold para que se agregue al listado de refactorings que gestiona. Cuando se reporta un *usability smell* de tipo *Forced bulk action* Kobold devolverá como sugerencia el refactoring *Distribute Menu* tal como se ve a continuación en el extracto de código:

```
[{
  "id" : 2,
  "name" : Distribute Menu,
   "description" : Add a link or button for each popular action to every item, so
   applying an action on a single element is simpler.,
  "additionalAttributes" : [{
              "description" : Checkbox XPath for each item on the list,
              "id" : 1,
              "type" : X,
              "label" : checkboxXPath
            },{
              "description" : Item's XPath,
              "id" : 2,
              "type" : X,
              "label" : contentXPath
            },{
              "description" : XPath to the button with the action to replicate,
              "id" : 3,
              "type" : X,
              "label" : buttonXPath
      }]
}]
```

A continuación se presenta un ejemplo de este refactoring aplicado por la extensión *Usability Smell Fixer.*

Como se mencionó anteriormente este refactoring apunta a la necesidad de dividir un conjunto de acciones específicas a un listado de elementos. En el ejemplo que presentamos a continuación nos enfocamos en el correo virtual www.gmail.com en su versión HTML. La razón por la cual se elige este sitio es porque, por ejemplo, para aplicar la acción de eliminar un correo o varios hay que marcar el *checkbox* del mail en cuestión y luego hacer click en el botón "Eliminar" que, como puede verse en la figura 6.3.4 se encuentra en el menú superior de la página, por lo que, si deseo eliminar varios elementos a la vez, debería marcar cada uno de ellos y luego subir para presionar dicho botón. Este escenario es muy complejo para las personas que poseen algún tipo de discapacidad motora o incluso personas no videntes.

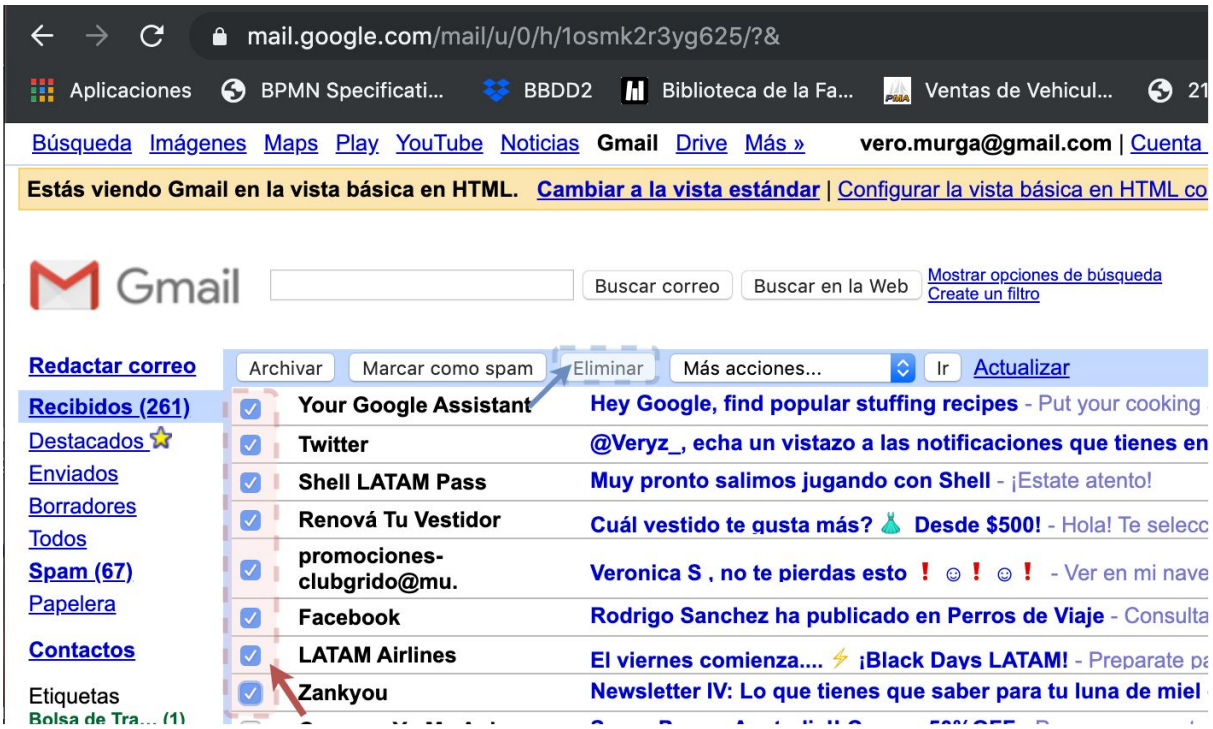


Figura 6.3.4. Identificación de conjunto elementos a los cuales se le aplica una misma acción.

En la figura 6.3.5 se puede observar los atributos completos que solicita la API Kobold para poder aplicar el refactoring correctamente.

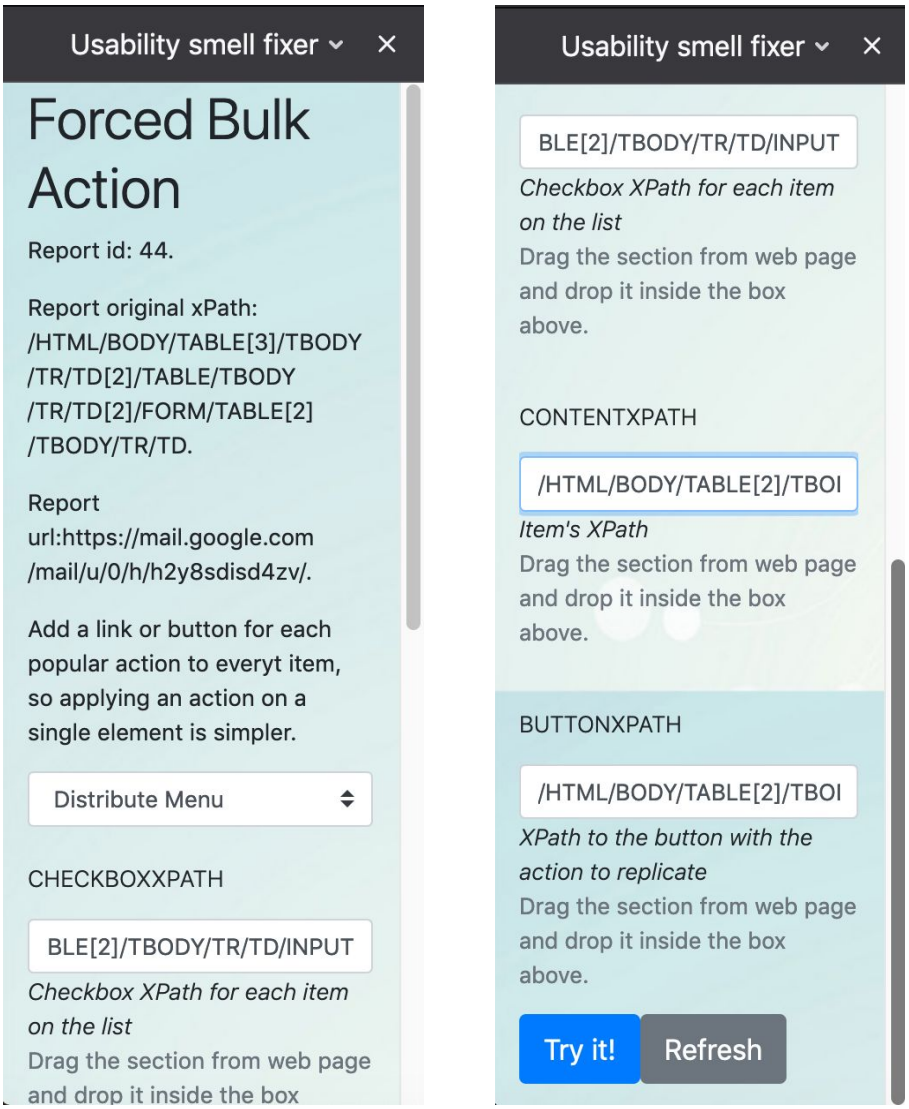


Figura 6.3.5. Carga de atributos para refactoring *Distribute Menu*.

Este refactoring necesita obligatoriamente tres componentes: el xPath del elemento que realiza la acción, en nuestro ejemplo el botón "Eliminar", el xPath al elemento donde se aplica la acción, en nuestro ejemplo el *checkbox* y el xPath al elemento donde se quiere reubicar la acción, en nuestro ejemplo al final de la fila del mail recibo. En la figura 6.3.6 se muestra a modo ejemplo la carga de los valores correspondientes.

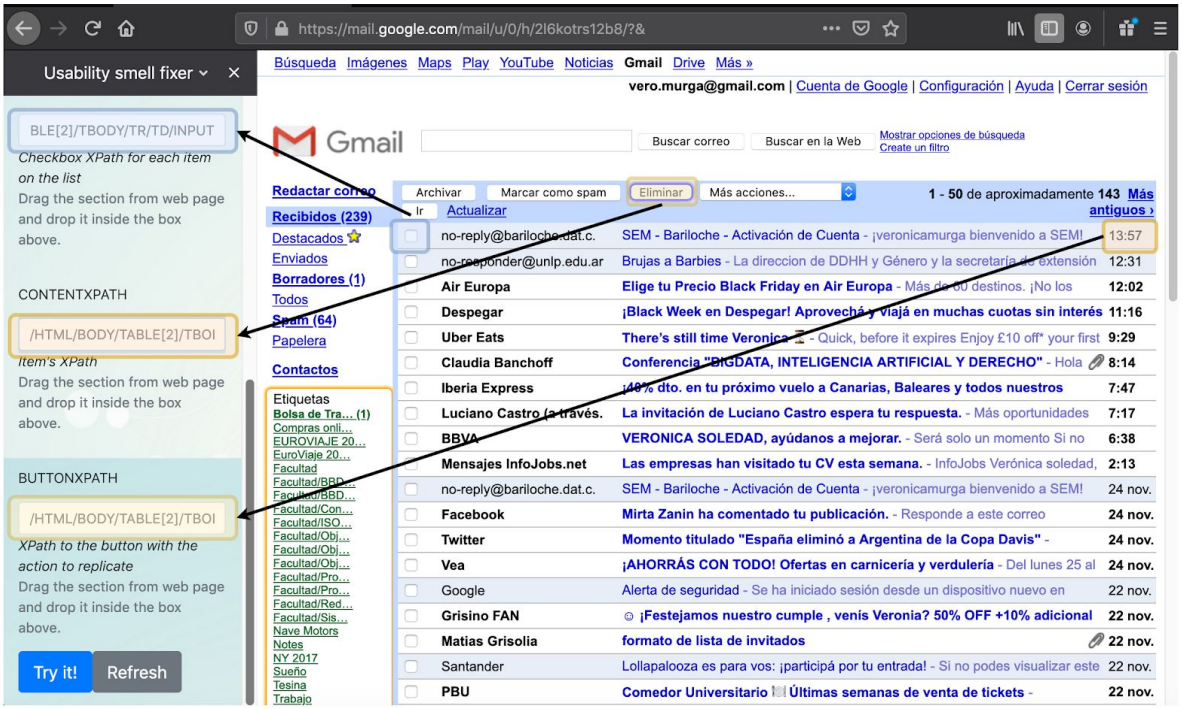


Figura 6.3.6. Selección de secciones para el refactoring. De color anaranjado se ve la selección de las acciones y de color azul el listado de elementos donde se aplicarán.

Una vez que se cargaron todos los valores obligatorios, se podrá ejecutar el refactoring presionando el botón "Try it!".En la figura 6.3.7, las acciones se distribuyen al final de cada elemento de la lista. Esto permite que el usuario no tenga que volver arriba de todo para poder ejecutar una acción global. Cada elemento posee su acción lo que permite que puedan ser ejecutadas individualmente.

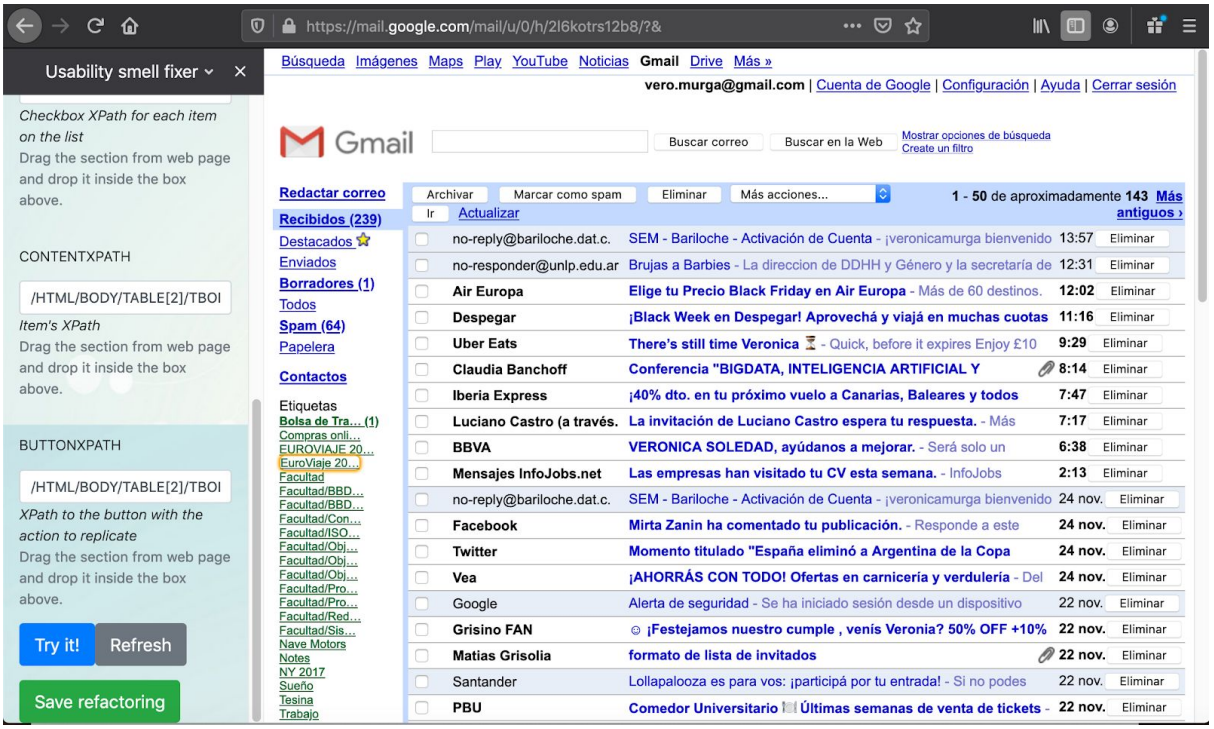


Figura 6.3.7. Refactoring Distribute menu ejecutado.

Este tipo de refactoring es de gran ayuda para aquellas personas con discapacidad visual o móvil. En el caso de los disminuidos visuales que utilicen screen reader para leer la página se verán muy beneficiados por este cambio, ya que no tendrán que volver para atrás para realizar las acciones. Los screen reader leen en orden los elementos de la página lo cual llevaría a que primero lean el checkbox, luego el contenido del correo y recién ahí el usuario sabría si quiere o no aplicar la acción, lo cual lo obligaría a volver atrás para seleccionar el checkbox y luego seguir subiendo hasta llegar a las acciones para aplicar a su selección.

CAPÍTULO 7

Casos de estudio

Las aplicaciones web hoy en día poseen un alcance inmenso. El ejemplo que abordaremos en este capítulo como caso de estudio serán los usuarios frecuentes y no tan frecuentes en los sitios webs de la prensa digital y página de emails. Los ingresos a estos sitios son cada día más demandantes ya que, tomando de ejemplo los diarios digitales, en estos tiempos es más frecuente leer el diario online que comprarlo en papel lo que provoca que los sitios web, si desean mantener la fidelidad de sus lectores, se vean obligados a mantenerse actualizados y permitir a todo tipo de usuario acceder a la información que publican, es decir, que el sitio sea fácil e intuitivo.

Caso 1. Prensa digital para usuarios no frecuentes

En este ejemplo utilizaremos el sitio web www.tn.com.ar. El usuario que realizará el reporte está pensando en usuarios nuevos en la navegación sobre internet y por lo tanto, nuevos en el sitio web mencionado, por ejemplo adultos de avanzada edad.

Íconos que pueden resultar confusos para usuarios no frecuentes

Al ingresar al sitio a primera vista se encuentra con algunos links que llevan a otras secciones y fotos con noticias pero inmediatamente nota la existencia de íconos que al pasar el mouse por arriba no explican su funcionalidad y esto podría provocar que aquellos usuarios que no son frecuentes en la web no entiendan y no puedan deducir la finalidad de dichos íconos. Es por ello que decide reportar el *usability smell* haciendo click en el ícono de la extensión en el margen superior derecho. En la figura 7.1.1 se podrá ver marcado este escenario.

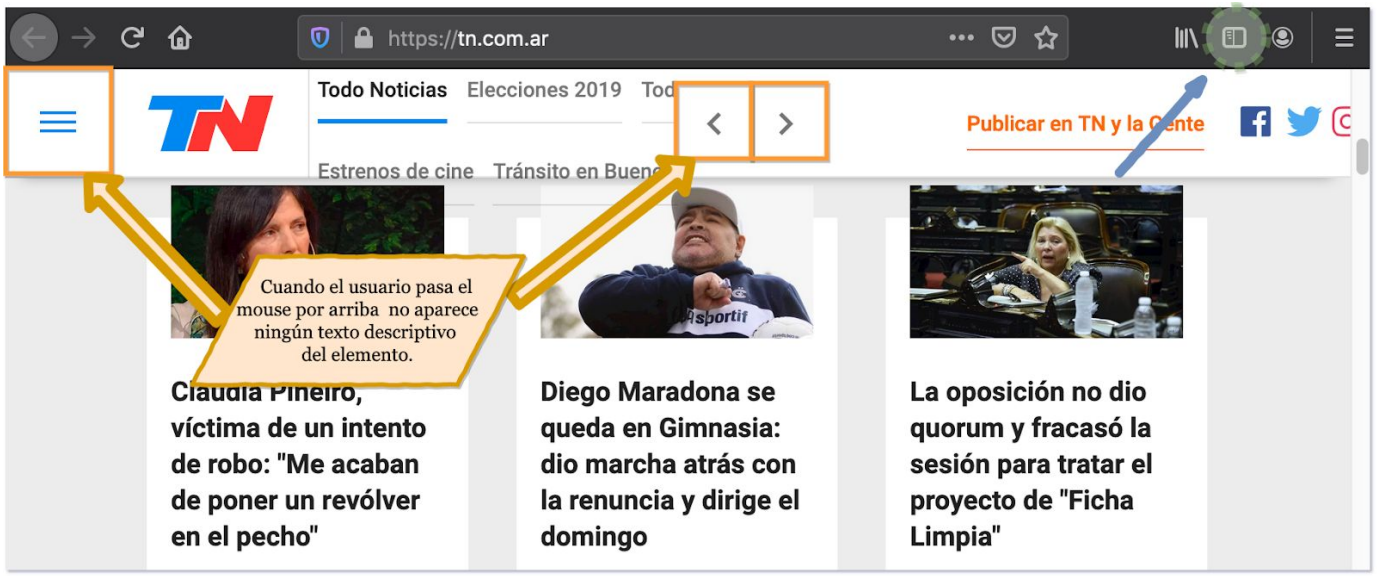


Figura 7.1.1. Página principal www.tn.com.ar. Se observan los BS detectados.

Cuando se abre la aplicación el usuario observa que aún no existe ningún refactoring asociado a esa página, ya que no figura ninguno en el listado de refactorings disponibles para la aplicación web actual (Figura 7.1.2) entonces decide habilitar el reporte para reclamar una solución al problema presionando en el botón *enable report*.

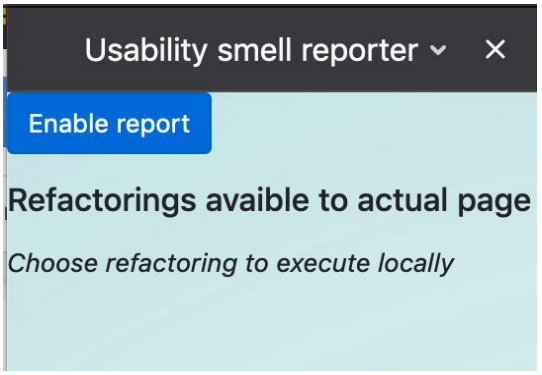


Figura 7.1.2. Leyenda que aparece cuando no existen refactorings asociados a la URL.

El usuario seleccionada la sección donde se presenta el problema y busca el usability smell indicado para este. En el caso de que no sea un experto en el área, tendrá disponible la descripción de cada uno de los BS para poder encontrar el indicado. El BS asociado al problema de usabilidad detectado es el *Undescriptive element* que indica que un elemento no es descriptivo por sí mismo. Esto ocurre usualmente cuando el usuario espera una descripción del elemento al pasar por arriba con el mouse. En la siguiente figura (7.1.3) se podrá observar cómo se completa el formulario del reporte.

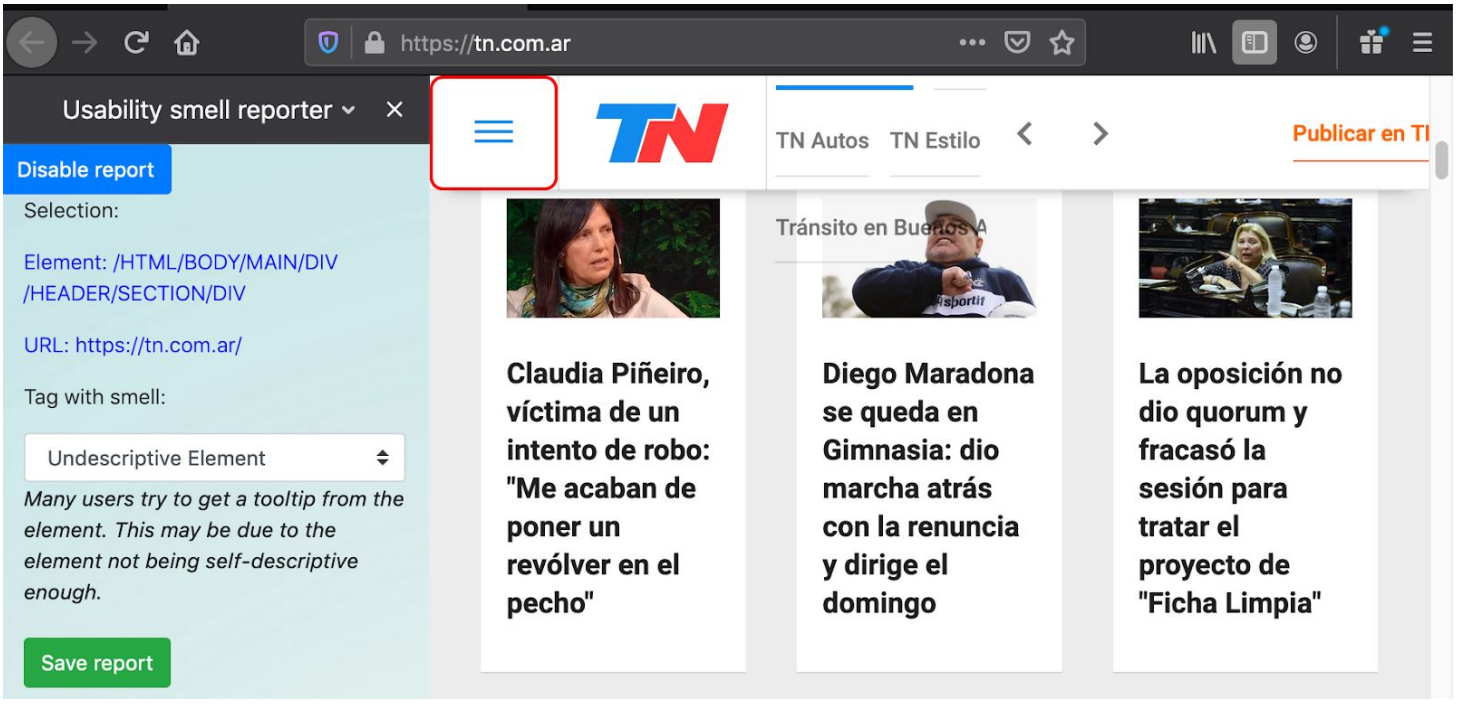


Figura 7.1.3. Formulario del reporte del BS *Undescriptive element.*

Luego de presionar el botón save report el usuario observa el mensaje de confirmación que se muestra en la figura 7.1.4. De esta forma el reporte ya queda guardado y listo para ser resuelto por el voluntario. Una vez guardado el reporte la aplicación queda lista para realizar otro reporte ya sea en la misma página o en otra diferente.

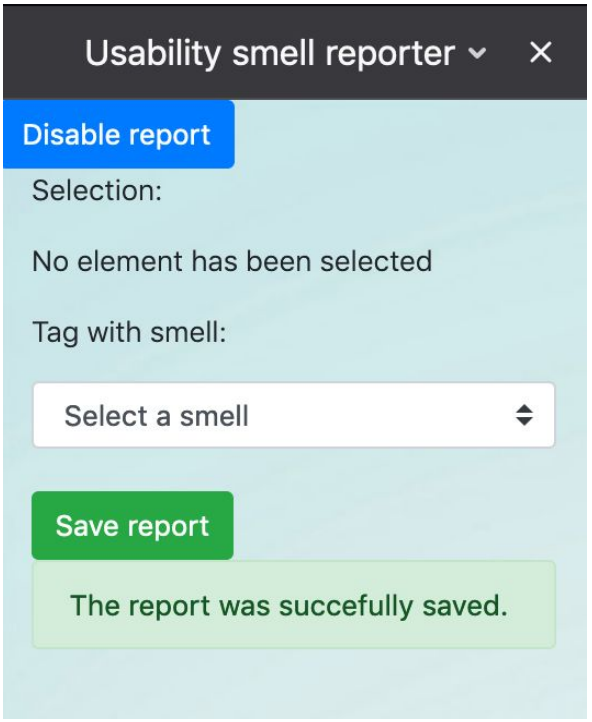


Figura 7.1.4. Mensaje de éxito de reporte.

El usuario repite este proceso dos veces más realizando un reporte por cada elemento marcado anteriormente en la figura 7.1.1. Al finalizar los reportes de BS el usuario queda a la espera de un voluntario que lo resuelva o puede ser él mismo el encargado de realizar la resolución utilizando las aplicaciones *Usability Smell Manager* y *Usability Smell Fixer*. Para continuar con nuestro ejemplo, el mismo usuario que realizó el reporte será el encargado de resolverlo.

Para resolver el reporte, el voluntario ingresa al sitio web donde se alojan estos a través de la aplicación *Usability Smell Manager* que se encuentra alojada en la dirección http://localhost:8001/smells/ y selecciona el reporte que desea solucionar. En la figura 7.1.5 se observan los tres reportes realizados anteriormente por el lector del diario online. El usuario selecciona el reporte número 40 y se lo redirige al sitio en cuestión y se carga la información sobre dicho reporte en la extensión *Usability Smell Fixer* para comenzar a buscar una solución al problema.

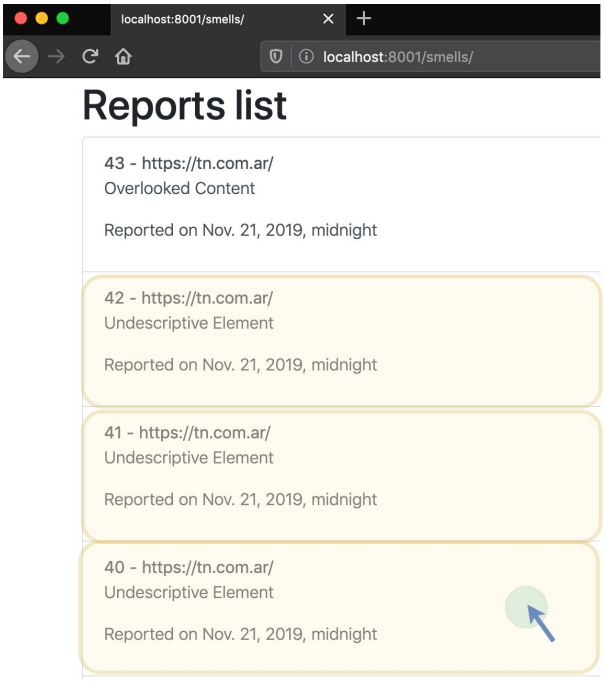


Figura 7.1.5. Usability Smell Manager

Para comenzar a resolver el BS el voluntario abre la extensión de navegador *Usability Smell Fixer* donde observa los detalles del reporte y un listado de refactorings sugeridos para resolver el problema tal como puede apreciarse en la Figura 7.1.6. En el ejemplo, para el *usability smell Undescriptive Element*, se sugieren dos refactorings: *Add ToolTip* y *Rename element*. El usuario no sabe con exactitud qué cambios ofrecen cada uno de estos refactorings por lo cual decide aprovechar la funcionalidad que le provee la herramienta: probar cada uno de los refactorings al vuelo y analizar cuál considera que es el más conveniente para resolver el problema reportado anteriormente.

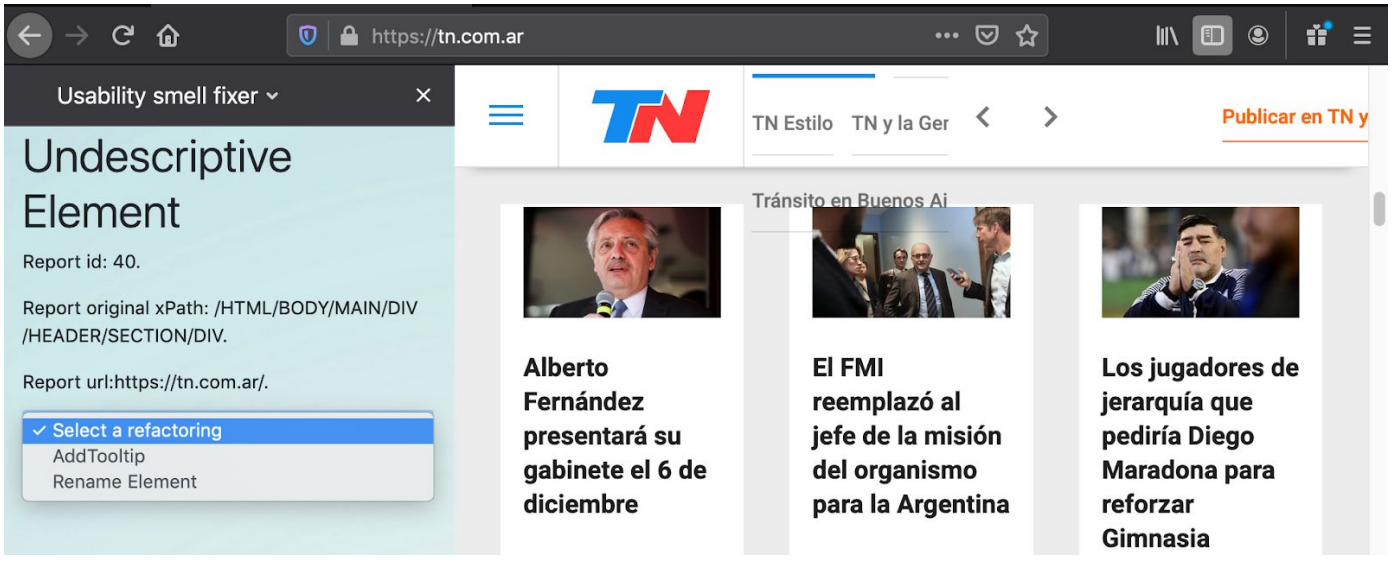
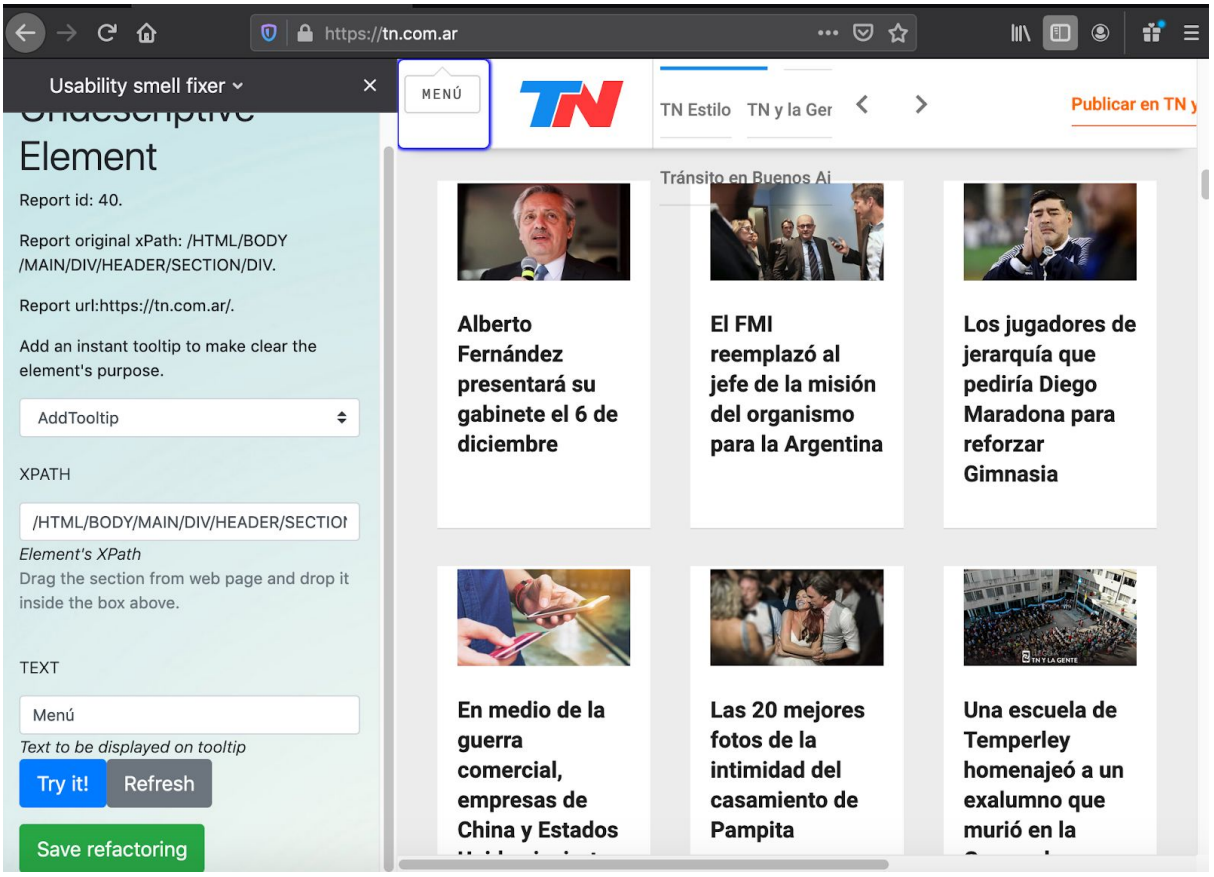


Figura 7.1.6. Estado inicial *Usability Smell Fixer*

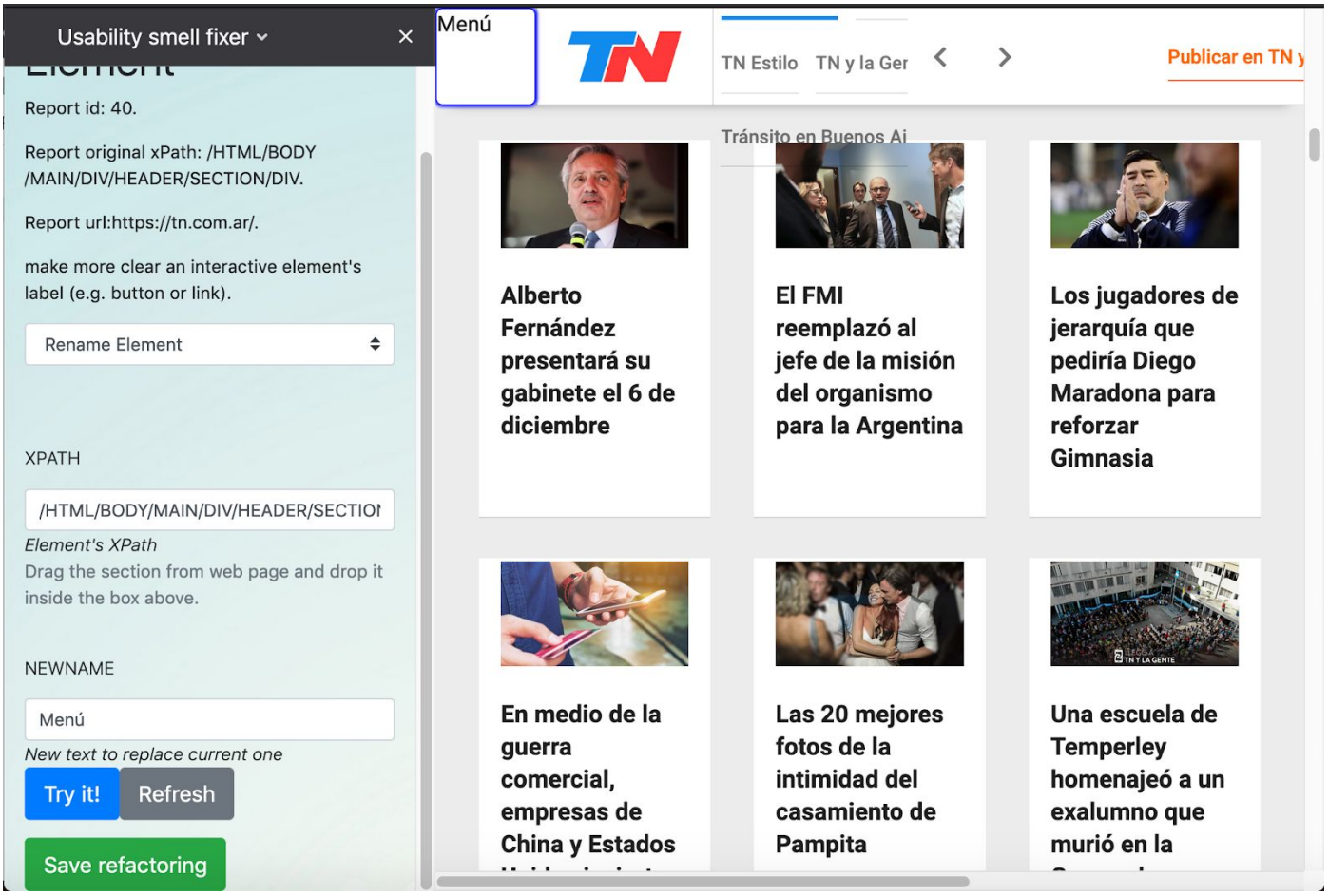
En la figura 7.1.7 se muestra la carga de refactoring addToolTip, el cual se encarga de agregar un tooltip en la zona reportada con el mensaje elegido por el usuario, en el ejemplo será "Menú". Para probar cómo se verá el refactoring aplicado, el voluntario presiona el botón "Try it!".



7.1.7. Reporte del botón Menú. Solución: add tooltip con el nombre "Menú". Luego de presionar el botón Try it! El refactoring se ejecuta y se agrega el tooltip al elemento seleccionado.

El refactoring se aplica y se puede observar que ahora al pasar por arriba del elemento con el mouse aparece el mensaje "Menú". El botón "Save refactoring" aparece en el caso de que el voluntario decida guardarlo y por lo tanto asociarlo al reporte.

En este caso el voluntario no está seguro si es el refactoring correcto, por lo que decide probar con la otra opción que se le ofrece. Para esto presiona el botón "Refresh" para quitar el refactoring ejecutado y elige el "Rename element" de la lista de refactorings ofrecida. Con este refactoring se renombra el elemento con el mensaje que indica el usuario. En la figura 7.1.8 se observa el formulario de resolución cargado y el botón "Try it!" presionado para ver cómo se ve aplicado el refactoring.



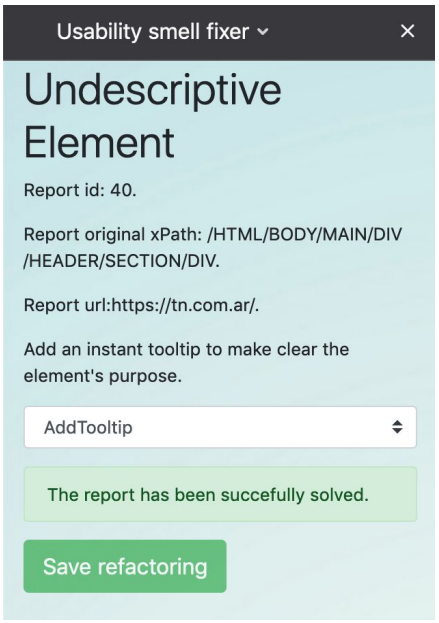
7.1.8. Reporte del botón Menú. Solución: rename element con el nombre "Menú". Luego de presionar el botón Try it! El refactoring se ejecuta y se modifica el elemento agregando el nuevo nombre elegido

Se puede observar ahora que el ícono se ha reemplazado por la leyenda provista: "Menú". En ambos casos cabe destacar que la funcionalidad del ícono sigue siendo la misma y que si se clickea en él se desplegará el menú original.

El voluntario finalmente se decide por la primera opción y guarda el refactoring AddToolTip en el reporte. Esta solución la repite dos veces más para los otros reportes realizados, agregando la leyenda "Siguiente" y "Atrás" en los elementos respectivos tal como se observa en la figura 7.1.9.

Figura 7.1.9. Resultado de reportes solucionados agregando leyenda "Siguiente"y "atrás" a los botones respectivos para que aparezca cuando el mouse se pasa por arriba.

Cuando termina de solucionar el reporte se verá la leyenda de éxito tal como se aprecia en la figura 7.1.10.



7.1.10. Refactoring guardado y por lo tanto asociado al reporte.

A partir de este momento los refactorings se encontrarán disponibles cuando el usuario ingrese al sitio en cuestión.

El usuario vuelve a la primera extensión Usability Smell Reporter y al ingresar al sitio www.tn.com.ar podrá obsevar la lista de refactorings sugeridos para el mismo. En este momento el usuario tendrá la posibilidad de seleccionar uno o varios refactorings para guardar localmente y de manera estos serán ejecutarlos cada vez que ingrese al sitio web. En la figura 7.1.11 puede verse el listado generado.

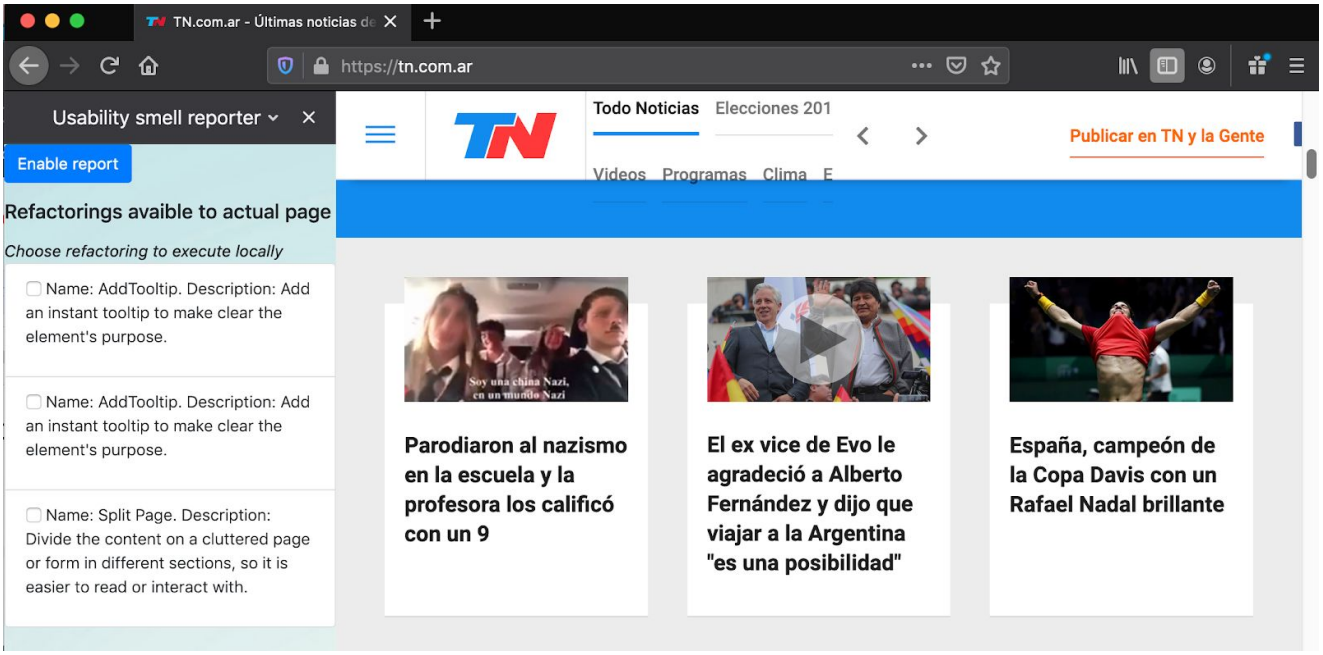


Figura 7.1.11. Listado de refactorings asociados al sitio web actual.

El usuario prueba el primer refactoring sugerido para ver que realmente se comporte como espera. En la figura 7.1.12 se puede observar que se agrega la leyenda "Atrás" al ícono que posee la flecha para retroceder.

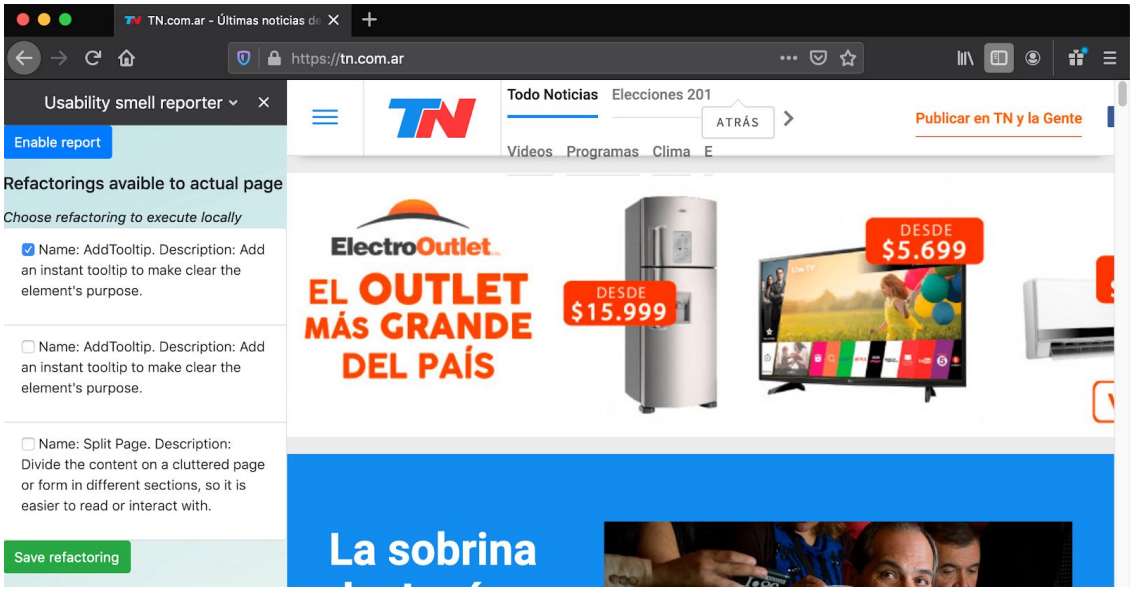


Figura 7.1.12. El usuario seleccionó el refactoring y observa el resultado en la página.

Como esta era la funcionalidad esperada por el usuario éste decide guardar el refactoring en su almacenamiento local para que de esta manera siempre que ingrese al sitio el refactoring se ejecute y cuente con esta mejora de usabilidad. Una vez presiona el botón "save refactoring" éste es guardado localmente. Para asegurarse de que todo salió bien el usuario espera el mensaje de éxito que se puede observar en la figura 7.1.13. Ahora cada vez que el usuario ingrese al sitio www.tn.com.ar éste refactoring se aplicará automáticamente y ya podrá gozar de sus beneficios. Si algún día el usuario decide eliminarlo simplemente tendrá que quitar el *checked* del checkbox y éste se eliminará de su almacenamiento local.

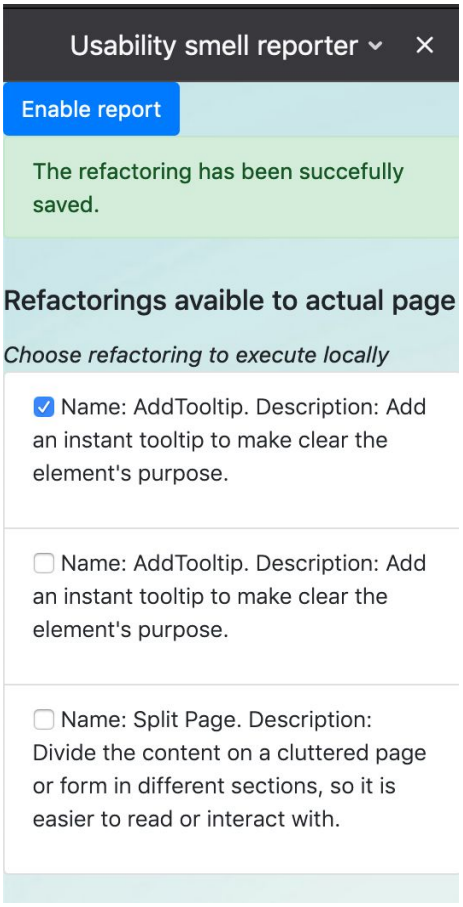


Figura 7.1.13. El usuario guarda el refactoring localmente.

Caso 2. Prensa digital para usuarios frecuentes

Así como hablamos de los usuarios nuevos en un sitio web ahora abordaremos aquel grupo de usuarios que ingresan a un determinado sitio web de manera frecuente. En el siguiente ejemplo volveremos a utilizar www.tn.com.ar como ejemplo.

Páginas con demasiado contenido usualmente ignorado por el usuario

En este caso el usuario observa que existe demasiado contenido en el sitio web y que la mayoría del mismo es ignorado por los usuarios frecuentes ya que éstos ingresarán la sitio y consumirán solo las secciones de su interés y el resto del contenido lo ignorarán. Al notar esto el usuario decide reportar el usability smell para ofrecer una solución a aquellos usuarios frecuentes. En la figura 7.2.1 puede observarse la zona del problema y que el usuario debió ir demasiado abajo en el sitio web para encontrar lo que buscaba. Abre la extensión Usability Smell Reporter presionando el ícono que se encuentra en el margen superior derecho para comenzar a realizar el reporte.

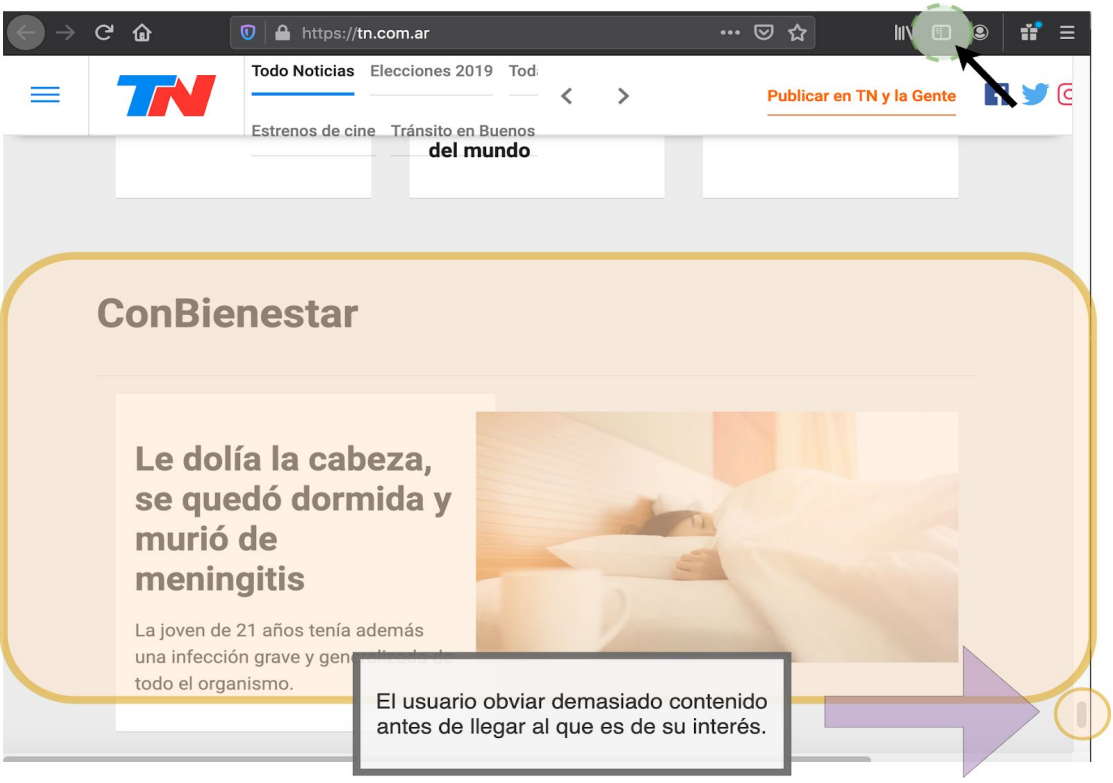


Figura 7.2.1. El usuario identifica demasiado contenido pasado de largo en el sitio web para llegar a un punto de interés específico.

Luego de buscar cuál es el BS ideal para este caso, el usuario se decide por reportar un *usability smell* de tipo *Overlook Content*. Este BS ocurre cuando existe contenido en un sitio web que es ignorado por el usuario. En la figura 7.2.2 se observa la selección del BS y el formulario de reporte completado.

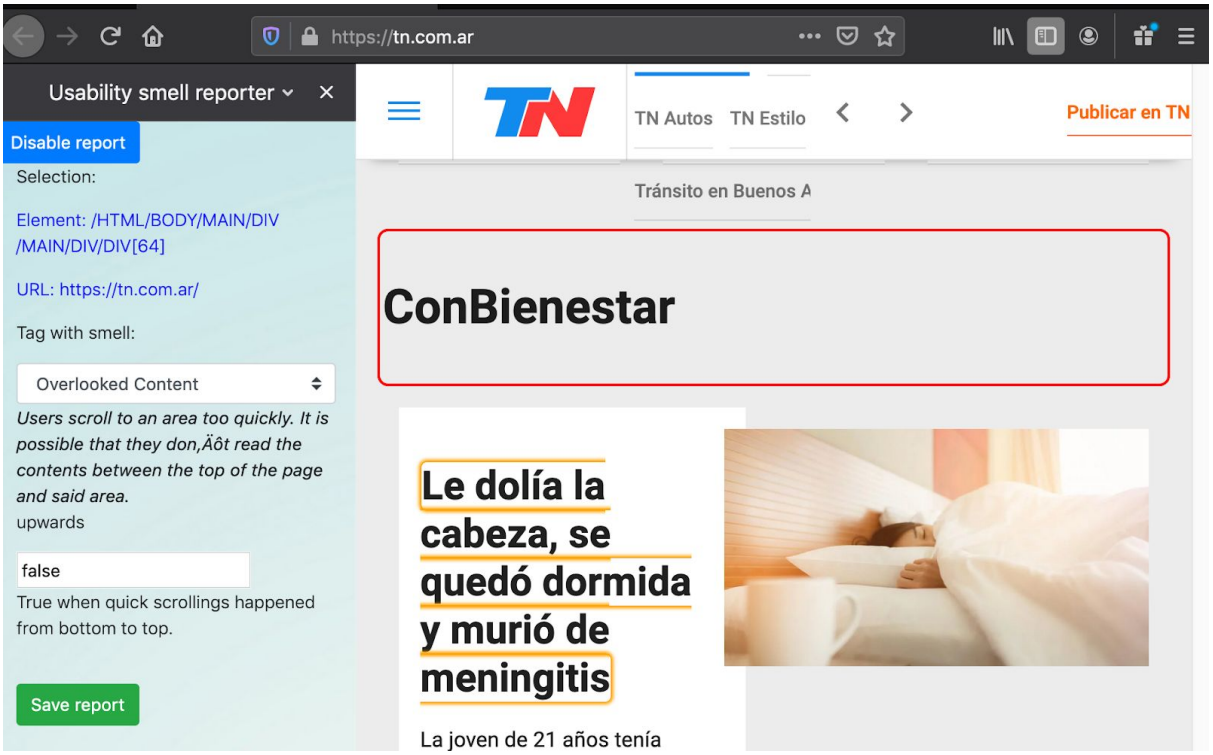


Figura 7.2.2. Formulario del reporte completado.

Una vez finalizado el reporte el usuario presiona el botón "save report" y lo guarda para que un voluntario lo solucione. En la imagen 7.2.3 se observa el mensaje de éxito al presionar dicho botón.

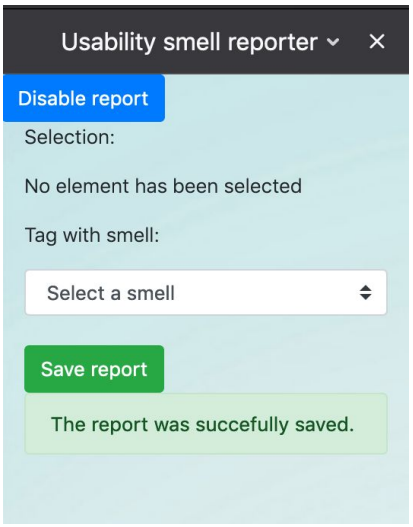


Figura 7.2.3. Reporte guardado con éxito.

Para solucionar el reporte el usuario ingresa al sitio web encargado de administrarlos. La url al sitio en nuestro ejemplo será http://localhost:8001/smells donde podrá verse el listado de reportes realizado. El usuario busca el que acaba de realizar y comienza a solucionarlo. Esta selección puede verse reflejada en la figura 7.2.4.

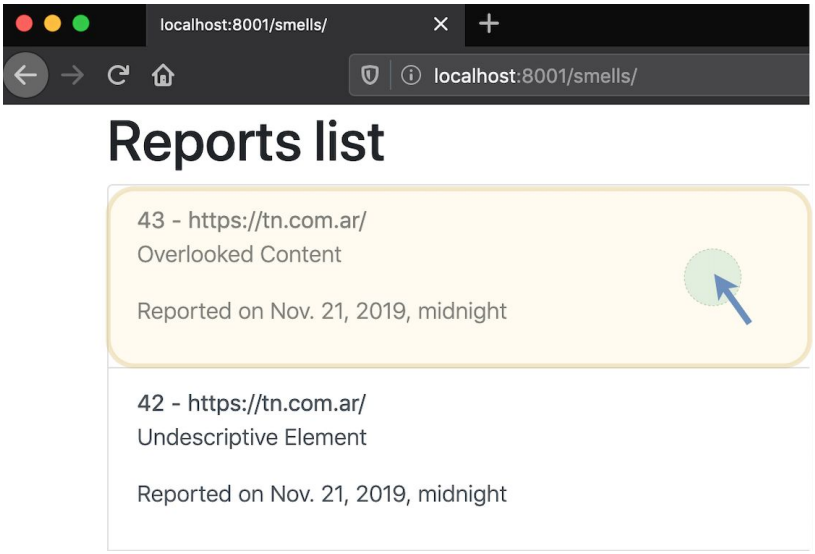


Figura 7.2.4. Selección del reporte en *Usability Smell Manager*.

Al presionar en el reporte se lo redirigirá al sitio en cuestión y al abrir la extensión verá el detalle del reporte correspondiente. También podrá observar el listado de refactorings sugeridos para solucionar el problema. En nuestro ejemplo el refactoring sugerido es el *Split Page*. Este refactoring sugiere dividir el contenido de la página en varias secciones. En la figura 7.2.5 se observa este escenario.

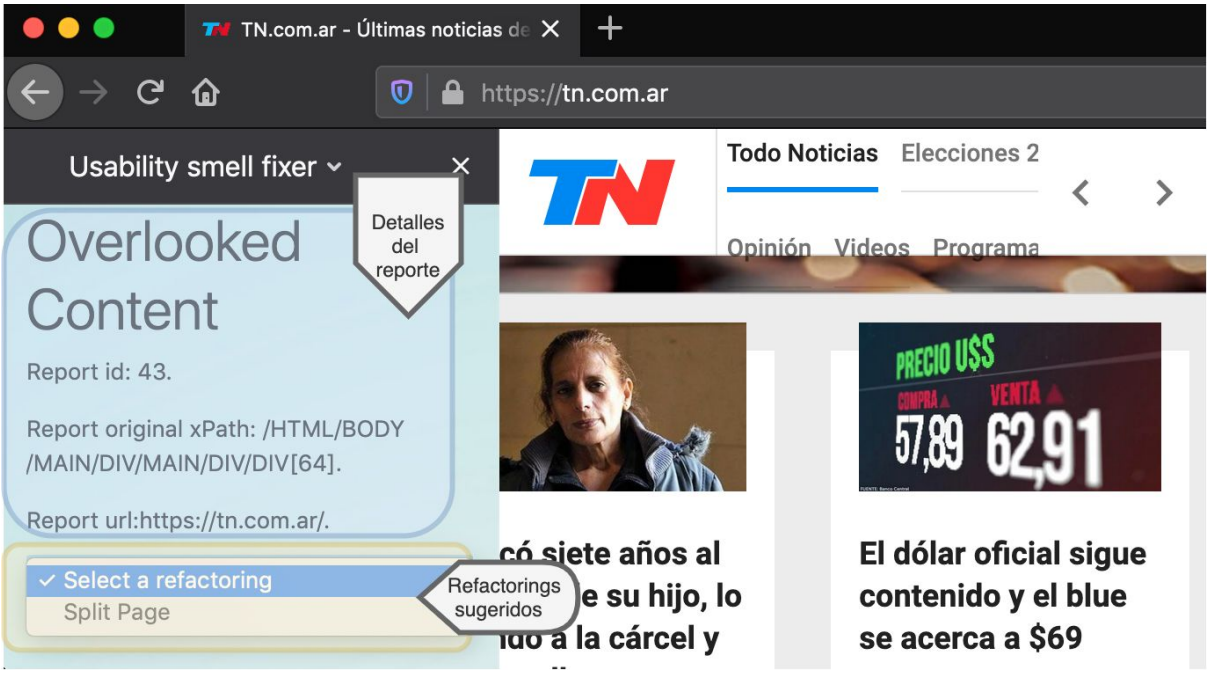


Figura 7.2.5. Usability Smell Fixer. Detalles iniciales del reporte a solucionar.

El usuario selecciona la opción del *Split Page* y aparecen dos secciones obligatorias para que éste complete. La primera corresponde a la sección principal (Main section) donde solo debe incluir el xPath a dicha sección. Luego le ofrece completar los datos de la

siguiente sección, pero en este caso es obligatorio incluir tanto el xPath a la sección como el nombre de la misma. En la figura 7.2.6 se ve cómo el usuario cargó los datos correspondientes a ambas secciones mencionadas.

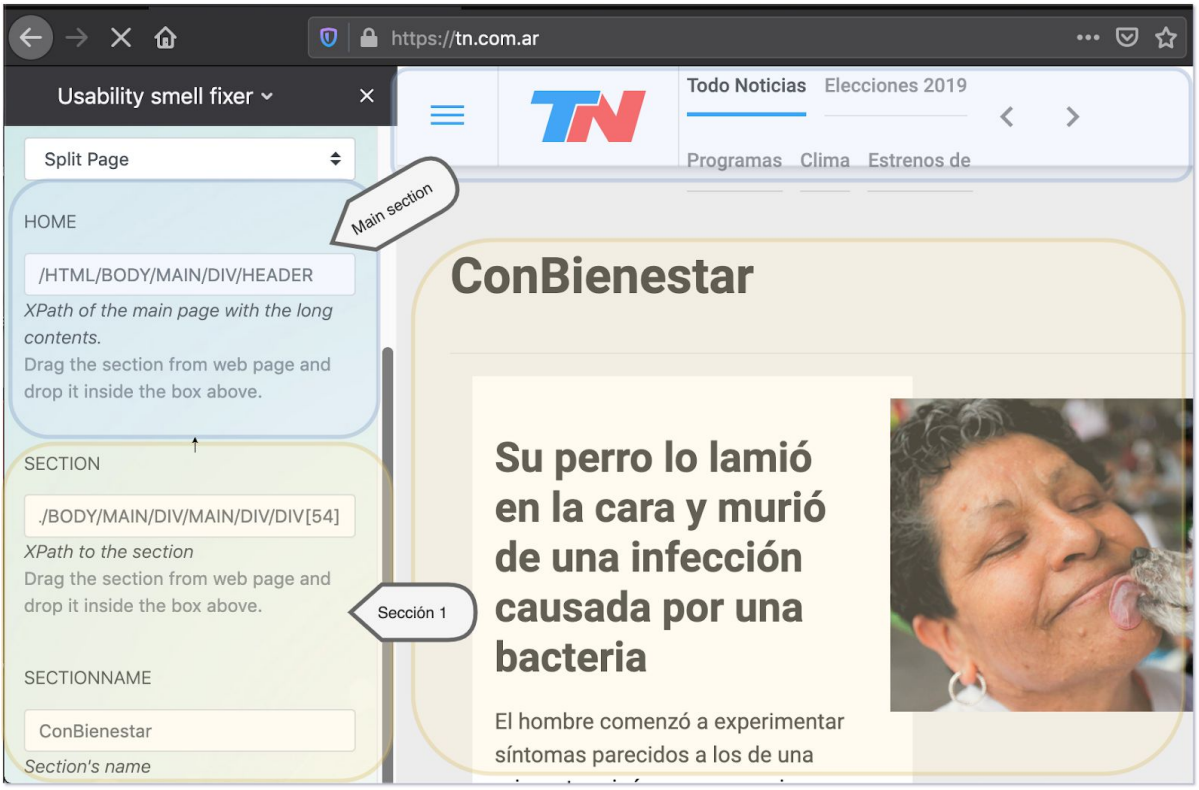


Figura 7.2.6. Split Page: selección de sección principal y de la primera sección.

En este caso para el usuario no son suficientes solo dos secciones, por lo que decide agregar una más presionando el botón "add" el cual le habilitará dos campos más para cargar el xPath y el nombre de la segunda sección (Figura 7.2.7).

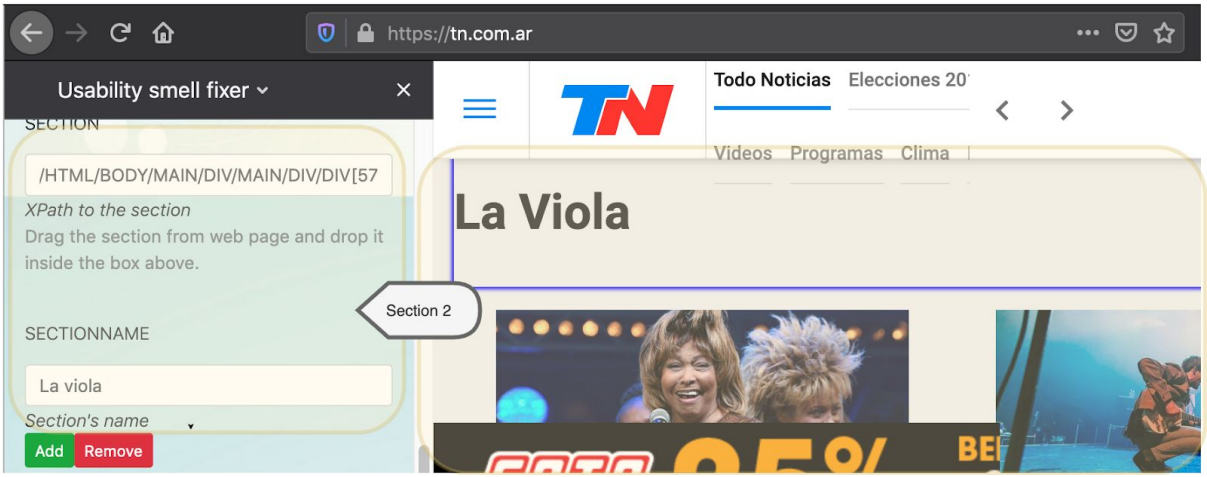


Figura 7.2.7. Split Page: selección de la sección segunda.

El usuario es libre de agregar o eliminar secciones a demanda, pero es obligatorio tener al menos la sección principal y la primera sección definida.

Una vez que cargó todos los datos prueba cómo va a quedar su solución presionando el botón "Try it!". En la siguiente figura (7.2.8) se puede observar cómo quedaría el refactoring ejecutado. En una primera instancia aparecerá la sección principal primero.

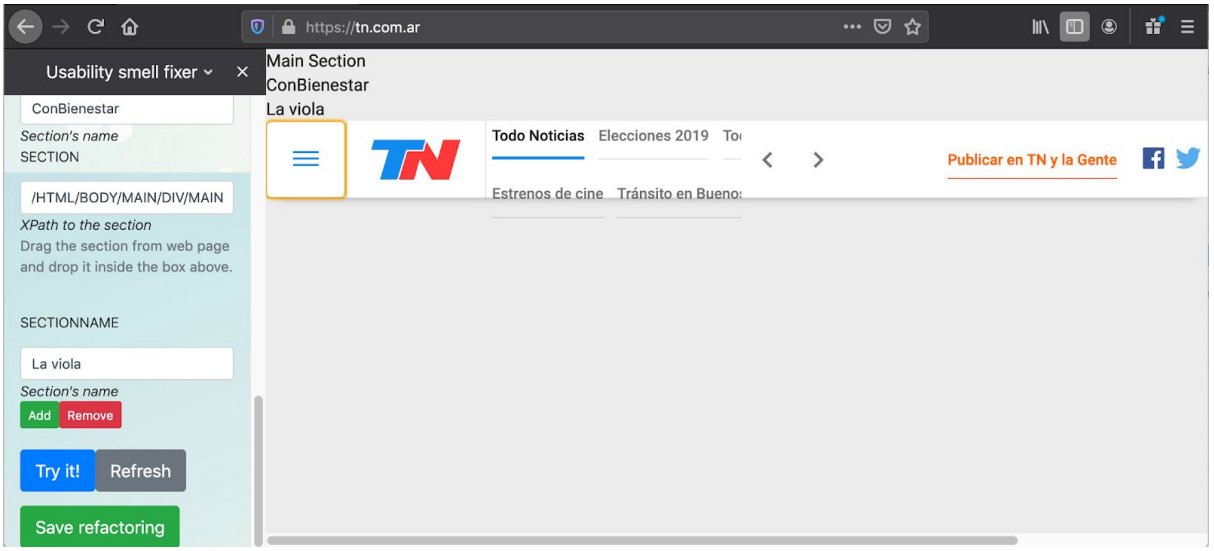


Figura 7.2.8. Split page: Try it! Imagen de sección principal.

Luego el usuario presiona la primera sección que definió "ConBienestar" y ve el resultado (Figura 7.2.9).

Figura 7.2.9. Split page: Try it! Imagen de sección primera.

Finalmente prueba la última sección que definió "La viola" y observa que es el resultado esperado (Figura 7.2.10). En el caso de que no sea el resultado esperado el

usuario puede cambiar los xPath y los nombres cuantas veces quiera e ir probando cómo se vería en el sitio.

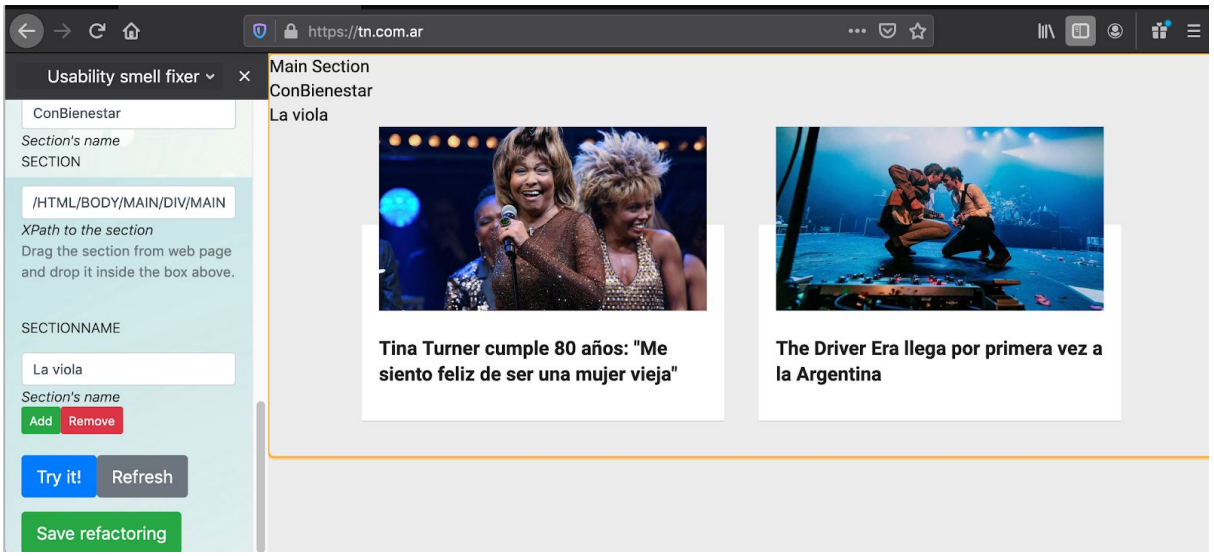


Figura 7.2.10. Split page: Try it! Imagen de sección segunda.

En nuestro ejemplo el usuario se encuentra satisfecho con los resultados por lo cual decide guardar el refactoring y de esa manera solucionar el reporte. En la figura 7.2.11 se ilustra este escenario.

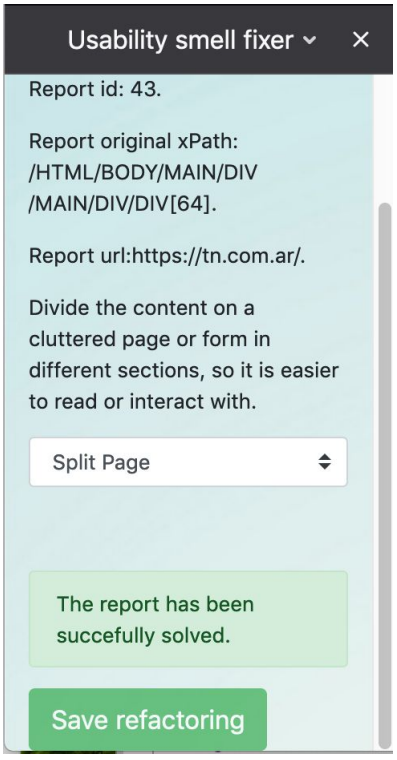


Figura 7.2.11. Reporte solucionado.

A partir de ahora cada vez que el usuario utilice la extensión Usability Smell Reporter e ingrese al sitio www.tn.com.ar verá sugerida esta solución. En la figura 7.2.12 veremos

que figuran dos resoluciones para el BS reportado *Overlook Content*. En este caso el usuario prueba ambas soluciones para ver cuál es la más conveniente y la que mejor se adapta a sus necesidades.

Figura 7.2.12. Usability Smell Reporter: aparece en el listado de reportes solucionados.

Luego de probar ambas el usuario define que la solución reflejada en la figura 7.2.13 es la mejor para él por lo cual decide guardarla en su almacenamiento local para que cada vez que ingrese al sitio www.tn.com.ar ésta se ejecute automáticamente.

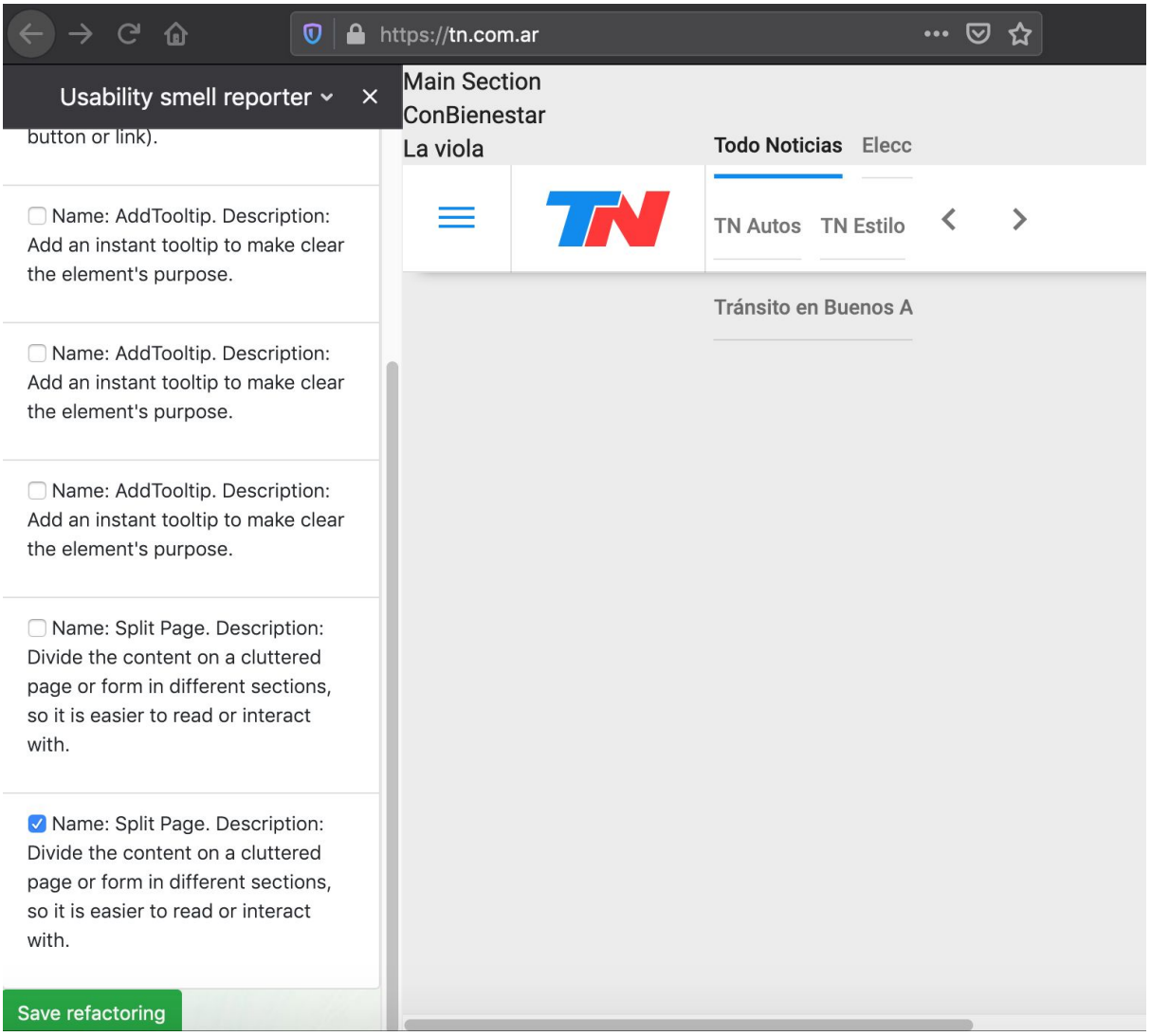


Figura 7.2.13. Split page ejecutado para probar su resultado.

El proceso finaliza con el mensaje de éxito confirmando que su operación fue realizada correctamente (figura 7.2.14). En el caso que el usuario se arrepienta y desee dar marcha atrás con su decisión podrá eliminar el refactoring de su almacenamiento local quitando el *click* del *checkbox*.

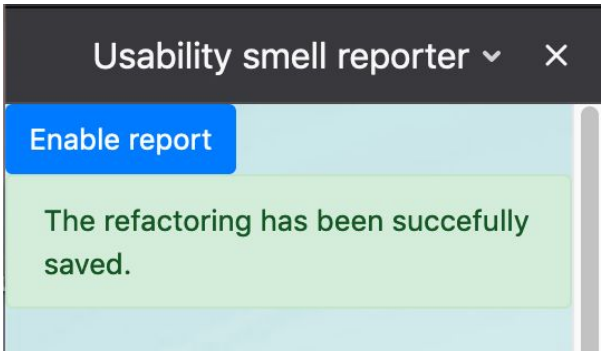


Figura 7.2.14. Split page guardado en el almacenamiento local.

CAPÍTULO 8

Conclusión y Trabajos futuros

A lo largo de este trabajo se destaca la importancia del concepto de usabilidad en las aplicaciones web para que los usuarios puedan utilizarlas sin dificultad. También se hace hincapié en que este concepto no es prioridad para las empresas a la hora de desarrollar un sitio web lo que puede provocar que los usuarios que los frecuentan decidan abandonarlos y buscar otras alternativas. Si bien algunas empresas se están comenzando a enfocar más en cuestiones de usabilidad debido a las exigencias de los usuarios, consideramos que no es suficiente para abarcar por completo las necesidades de todos. Por ejemplo los usuarios no videntes no tienen las mismas necesidades que un usuario vidente pero novato en cuestiones de navegación web. En este trabajo decidimos enfocarnos en todos los tipos de usuarios y ofrecer soluciones en conceptos de usabilidad dándole la oportunidad a los usuarios de aplicar los cambios que consideren necesarios para mejorar su experiencia en el sitio web sin la necesidad de tener conocimientos técnicos informáticos ni de depender de los dueños del sitio web en cuestión. Logramos aislar los términos técnicos ofreciéndole al usuario una interfaz amigable para interactuar.

Se describieron varias herramientas que permiten al usuario mejorar la usabilidad en las aplicaciones web aplicando los cambios del lado cliente web mediante la ejecución de refactorings. Algunas desventajas de estas herramientas es que exigen que los usuarios posean conocimientos técnicos o no permiten que los usuarios especifiquen cuál es el problema para aplicar la mejora a la necesidad de cada uno. Nuestra herramienta permite que usuarios novatos o expertos puedan sugerir y mejorar la usabilidad de los sitios webs con el objetivo de satisfacer las necesidades de los usuarios para que no se sientan frustrados a la hora de navegar por sus sitios favoritos.

Una de las cuestiones más destacadas a lo largo de este trabajo es la ayuda de la comunidad centrándonos en el concepto de crowdsourcing. Este método es muy potente ya que permite que varios voluntarios presenten una solución a un mismo reporte dejando al usuario elegir la que mejor se adapta a su necesidad. Es muy importante contar con la ayuda de los voluntarios a la hora de realizar los reportes o de solucionar los mismos ya que estos realizarán el aporte más importante enriqueciendo la comunidad. Nuestra herramienta permite participar a todos los usuarios que lo deseen sean o no usuarios frecuentes de los sitios reportados o sean o no técnicos informáticos. Es posible que el mismo usuario que realiza el reporte sea el que lo soluciona, permitiendo a éste realizar los cambios que espera, en términos de usabilidad, de un sitio web y de esta forma adaptarlo a su necesidad.

Teniendo en cuenta todos los aspectos mencionados anteriormente se desarrolló Mellito. Éste está compuesto por tres aplicaciones, dos extensiones de navegador y una aplicación web. Utilizando la extensión de navegador *Usability Smell Reporter*, los usuarios podrán de forma intuitiva realizar un reporte de un problema de usabilidad que puedan llegar a encontrar en cualquier aplicación web sin necesidad de ser un experto informático. Esto se da gracias a los formularios desarrollados que poseen descripciones suficientes para poder catalogar al reporte y la funcionalidad *drag and drop* que posibilita la selección del elemento en la página sin dificultad. Esto ayuda a que tanto los usuarios frecuentes del sitio, que deseen mejorar su experiencia sobre el mismo, como los usuarios ocasionales, que simplemente desean reportar un problema de usabilidad en éste, puedan realizar los reportes sin dificultad y de esta manera mejorar la usabilidad de la aplicación web para cualquier usuario que desee gozar de estos cambios.

La aplicación web *Usability Smell Manager* y la extensión *Usability Smell Fixer* trabajan en conjunto para resolver los reportes realizados por los usuarios. Los voluntarios que decidan solucionar los reportes no deberán ser necesariamente expertos informáticos ya que la herramienta provee una funcionalidad que les permite probar las posibles soluciones sugeridas en términos de *usability refactoring*. Siendo esto una gran ventaja para ampliar la comunidad de voluntarios con aquellos que no son técnicos informáticos y permitiendo también a usuarios que quieran solucionar sus propios reportes para adaptarlos de esta manera a su necesidad.

Utilizando como apoyo el trabajo realizado en *Kobold: web usability as a service* [Grigera17b] se desarrolló una API para comunicarse con Kobold y de esta manera independizar a nuestra herramienta de la manutención de los *usability smells* y de los *usability refactorings*, permitiendo centralizar la información en un solo sitio haciéndolo más mantenible ante cualquier tipo de cambio. De esta forma se utiliza el estándar de comunicación provisto por la API Kobold y se solicita la información necesaria sobre los *usability smells* y los *usability refactorings* para poder realizar el flujo de reporte y resolución de BS con éxito.

Teniendo en cuenta que existen distintos tipos de usuarios con diferentes necesidades sobre un mismo sitio, se desarrolló el refactoring *Distribute Menu* y *Split Page* como aporte a los ya existentes en Kobold. Por ejemplo si tenemos en cuenta a usuarios no videntes que utilizan un lector de pantalla para navegar por internet, les sería muy difícil manipular algunos sitios web, como por ejemplo el correo electrónico *gmail* tal como presentamos como ejemplo en este trabajo. Como demostramos aplicando el refactoring *Distribute Menu* logramos mejorar la experiencia de estos usuarios para que no tengan que realizar demasiados pasos para, por ejemplo, eliminar un mail. Una persona sin visibilidad disminuida podría tranquilamente realizar sin ningún esfuerzo la acción de hacer click en el *checkbox* y luego presionar el botón eliminar que se encuentra arriba de todo. Continuando con el mismo ejemplo de un usuario no vidente una página con contenido extenso podría ser dificultosa para navegar sobre todo si el usuario solo quiere acceder a un determinado conjunto de información. Como solución a este problema se desarrolló el algoritmo *Split*

Page para que una página pueda ser dividida en secciones y de esta manera mejorar la experiencia del usuario al navegarla.

Como trabajo futuro se propone la ampliación de la aplicación Usability Smell Manager para que soporte el login de usuarios y de esta manera poder proteger la integridad de los reportes. Definir roles para que existe un administrador general que pueda modificar todos los reportes y limitar a los roles de usuarios voluntarios para que solo puedan solucionar los reportes sin realizar mayores alteraciones sobre estos.

Actualmente las aplicaciones fueron escritas en inglés como idioma base para poder ser utilizada de forma internacional. Se propone desarrollar la internacionalización de las mismas para que se adaptan al idioma de cada usuario según el país.

Una de las desventajas que se presentan es el nivel de ejecución de los refactorings, siendo este necesario para que los refactorings no se solapen y que sigan comportándose de la forma esperada. Se propone como trabajo futuro definir diferentes niveles de ejecución y por lo tanto distintos niveles de reportes ya que habrá que tener en cuenta que los xPath cambiarán luego de haberse ejecutado un refactoring específico.

Adicionalmente agregar la funcionalidad para permitir al usuario técnico informático desarrollar refactorings personalizados para solucionar un determinado *usability smell* y tener la posibilidad de subirlos y compartirlos con la comunidad.

Finalmente se proponen mejoras visuales para el guardado y la presentación de los reportes solucionados para que no exista confusión a la hora de elegir uno siendo que, actualmente solo se especifica el nombre del refactoring y la descripción por lo que el usuario deberá probar ambos refactorings para saber cuál es el que desea.

Referencias bibliográficas

[Brabham14] Daren C. Brabham. Crowdsourcing as a Model for Problem Solving An Introduction and Cases, University of Utah, USA. 2014.

[Brhel15] Brhel, M., Meth, H., Maedche, A. Werder, K. "Exploring principles of user-centered agile software development: A literature review". Information and Software Technology 61 (2015) 163-181.

[Brooke96] Brooke, J. (1996). SUS: A "quick and dirty" usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & A. L. McClelland (Eds.), Usability Evaluation in Industry. London: Taylor and Francis.

[Carta11] Tonio Carta, Fabio Paternò, and Vf De Santana. 2011. Web usability probe: a tool for supporting remote usability

evaluation of web sites. In Human-Computer Interaction – INTERACT 2011. LNCS 6949. 349–357.

[Django05] 2005-2019 Django Software Foundation and individual contributors. Django is a registered trademark of the Django Software Foundation - https://docs.djangoproject.com

[Fernandez11] Fernandez, A., Insfran, E., & Abrahão, S. (2011). Usability evaluation methods for the web: A systematic mapping study. Information and Software Technology, 53(8), 789–817.

[Fowler99] M. Fowler. Refactoring: Improving the design of existing code. Addison-Wesley 1999.

[Garrido11] A. Garrido, G. Rossi, and D. Distante, "Refactoring for Usability in Web Applications.," IEEE Softw., vol. 28, no. 3, pp. 60–67, 2011.

[Garrido13a] Garrido, A., Firmenich, S., Rossi, G., Grigera, J., Medina-Medina, N., y Harari, I. Personalized Web Accessibility using Client-Side Refactoring. IEEE Internet Computing 17 (4), 2013.

[Garrido13b] Garrido, A., Rossi, G., Medina-Medina, N., Grigera, J., Firmenich, S. Improving Accessibility of Web interfaces: refactoring to the rescue. Univ. Access in the Information Society 2013.

[Grigera17a] J. Grigera, A. Garrido, J. M. Rivero, and G. Rossi, "Automatic detection of usability smells in web applications," Int. J. Hum. Comput. Stud., vol. 97, pp. 129–148, 2017.

[Grigera17b] J. Grigera, A. Garrido, G. Rossi. "Kobold: web usability as a service". ASE 2017: 990-995

[Howe08] Jeff Howe. 2008. Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business (1 ed.). Crown Publishing Group, New York, NY, USA.

[ISO11] ISO, I (2011) ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements

and Evaluation (SQuaRE) - System and software quality models.

[Krug05] Krug, S. Dont make me think, Segunda edición. Addison Wesley 2005

[Ko11] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck, "The state of the Art in End-User Software Engineering". ACM Computing Surveys, Vol. 43, No. 3, Article 21. 2011.

[MDN18] MDN web Docs - moz://a (2005-2018). Mozilla y colaboradores individuales. - https://developer.mozilla.org/es/ - 16/05/2018

[Nielsen90] Nielsen, J., and Molich, R. (1990). Heuristic evaluation of user interfaces, Proc. ACM CHI'90 Conf. (Seattle, WA, 1-5 April) 249-256.

[Nielsen99] Nielsen, 1999. Designing web usability: The practice of simplicity.

[Nielsen06] Nielsen, J., Loranger, H., 2006. In: Peri, C. (Ed.), Prioritizing Web Usability. Pearson Education.

[Sullivan00] Sullivan, T., & Matson, R. (2000, November). Barriers to use: usability and content accessibility on the Web's most popular sites. In Proceedings on the 2000 conference on Universal Usability (pp. 139-144). ACM.

[Takagi08] Hironobu Takagi, Shinya Kawanaka, Masatomo Kobayashi, Takashi Itoh, and Chieko Asakawa. 2008. Social accessibility: achieving accessibility through collaborative metadata authoring. In Proceedings of the 10th int. ACM SIGACCESS conf. on Comp. and accessibility. ACM, New York, NY, USA, 193-200.

[W3C] World Wide Web Consortium. https://www.w3.org/. Accedido el 15/11/19

Índice de figuras

Anexo A

API Usability Smell

Solicitar listado de reportes

Devuelve el listado de los reportes realizados por los voluntarios desde la extensión *Usability smell reporter*. Este listado incluye el *smell* y los valores de los atributos en el caso de que se hayan cargado.

Request

```
GET /smells/api/reports/
```

Response ejemplo

```
[{
   "id": 22,
   "href": "http://www.example.org/", ← URL donde se encuentra el smell
  "xpath": "/HTML/BODY/DIV", ← Elemento asociado al lugar específico del reporte
   "date": "2019-06-04T00:00:00-03:00",
   "idSmellKobold": 6, ← id para identificar el smell en Kobold
   "description": "Free Input for Limited Values", ← Smell asociado
   "attributes": [{ ← Valores de los atributos del smell asociado
        "value": "valores frecuentes",
        "idKobold": 2 ← id para identificar el atributo en Kobold
       },{
        "value": "valores 2",
        "idKobold": 2
       }]
}]
```

Solicitar smell report por ID

Devuelve el reporte por ID. Corresponde al reporte realizado por un usuario desde la extensión *Usability smell reporter.* Incluye el *smell* y los valores de los atributos en el caso de que se hayan cargado.

Request

```
GET /smells/api/reports/[idReport]
```

Response ejemplo

```
{
   "id": 22,
```

```
   "href": "http://www.example.org/", ← URL donde se encuentra el smell
  "xpath": "/HTML/BODY/DIV", ← Elemento asociado al lugar específico del reporte
   "date": "2019-06-04T00:00:00-03:00",
   "idSmellKobold": 6, ← id para identificar el smell en Kobold
   "description": "Free Input for Limited Values", ← Smell asociado
   "attributes": [{ ← Valores de los atributos del smell asociado
        "value": "valores frecuentes",
        "idKobold": 2 ← id para identificar el atributo en Kobold
       },{
        "value": "valores 2",
        "idKobold": 2
       }]
}
```

Recuperar refactorings por id de reporte

Lista de refactorings con sus atributos. Estos corresponden a los atributos obligatorios y sus valores que se completan para mandar a *Kobold* y de esta forma devuelva el código javascript del refactoring a ejecutar.

La información extra de los refactorings (nombre, descripción) se cargan en la base de datos a modo de backup cuando se soluciona un reporte, utilizando la información que envía Kobold al pedir el listado de refactorings para determinado smell. No debería usarse en la aplicación *Usability smell fixer* para recuperar el listado de refactorings ya que el flujo normal sería mandar a Kobold el *smell* y la lista de atributos para que este nos devuelva la lista de refactorings y sus atributos. El único fin que cumple este endpoint es el de poder nutrirse de la información necesaria para solicitar a kobold el código del refactoring por id_kobold.

Request

```
GET /smells/reports/[idReport]/refactorings
```

Response ejemplo

```
[{
  "id": 75,
  "name": "InputIntoRadios", ← Refactoring que se seleccionó para el reporte
  "description": "Turn a regular text field into a set of radio buttons with an
other option.",
  "id_kobold": 10, ← ID del refactoring en Kobold
  "attributes": [{
       "id": 45,
       "idKobold": 1,
       "date": "2019-11-17T17:55:46.030380-03:00",
       "value": "/HTML/BODY/HEADER/DIV/DIV/NAV/UL[2]/LI/DIV/FORM/INPUT",
        "attr_type": "S"
      }],
   "smellReport": {
       "id": 35,
```

```
       "href": "https://www.icann.org/resources/pages/idn-2012-02-25-en",
       "xpath": "/HTML/BODY/HEADER/DIV/DIV/NAV/UL[2]/LI/DIV/FORM/INPUT",
       "idSmellKobold": 6
     }
}]
```

Guardar reporte

Guarda el reporte realizado por un voluntario desde la extensión *Usability smell reporter.* Como parámetro se deberá enviar toda la información necesaria del refactoring seleccionado para resolver el reporte.

Request

```
POST /smells/saveReport
```

Body ejemplo

```
{
  "xPath": "/HTML/BODY/DIV", ← xPath donde se encuentra el smell
  "url" : "http://www.google.com/recurso", ← URL afectada
  "smellId" : "6", <- ID del smell reportado (corresponde al ID en Kobold)
  "description" : "Free Input for Limited Values"
  "smellAttributes": [{
         "id": 28,<- ID del atributo (corresponde al ID en Kobold)
         "value": "Valor frecuente 1"
       },{
         "id": 28,<- ID del atributo (corresponde al ID en Kobold)
         "value": "Valor frecuente 2"
       }]
}
```

Response

```
HTTP status 200 OK
```

Recuperar la lista de reportes solucionados por URL

Recupera la lista de refactorings aplicados como solución a un reporte realizado para la URL enviada por parámetro. Dicho listado traerá los datos necesarios sobre cada refactoring para luego pedirle a Kobold el código javascript para poder ejecutarlo.

Request

```
POST /smells/refactoringsByUrl
```

Body *ejemplo*

{ "url" : "http://www.example.org" }

Response *ejemplo*

```
[{
  "id": 17,
  "name": "Split page",
  "description": "Divide la página en subsecciones relacionadas entre sí",
  "id_kobold": 8,
  "attributes": [{
        "id": 28,
         "idKobold":1,
         "attr_type": "X",
         "date": "2019-01-21T20:27:52-03:00",
         "value": "direccion"
      },
      [{
        "id": 28,
         "idKobold":2,
         "attr_type": "X",
         "date": "2019-01-21T20:27:52-03:00",
         "value": "xPathValue"
        }]
     ]
}]
```

Guardar refactoring para reporte

Guarda el refactoring seleccionado como solución al reporte. Los parámetros se envían en el cuerpo del request. El refactoring se guarda con los atributos, en el caso que corresponda, y sus valores. Se asocia al reporte en cuestión.

Request

```
POST /smells/refactorings
```

Body *ejemplo*

```
{ "currentReportId": "9", ← ID del reporte que se está solucionando
  "koboldId" : "1", ← ID del refactoring en Kobold
  "attributes": [{
        "id": 28,
        "label": "Main section",
        "description": "xPath de la sección principal",
        "attr_type": "X",
        "date": "2019-01-21T20:27:52-03:00",
        "value": "direccion"
       },{
                "id": 29,
                "label": "New section",
```

```
                "description": "Nombre de la sección",
                "attr_type": "T",
                "date": "2019-01-21T20:29:21-03:00",
                "value": "Otra"
            }]

}
```

Response

```
HTTP status 201 OK.
```

Solicitar lista de smells

Devuelve el listado de smells disponibles con sus atributos. Estos atributos no son obligatorios, pero si el usuario los completa le permite a Kobold devolver un listado de refactorings más específico para solucionar el problema.

Request

```
GET /smells/api/smells
```

Response

```
[{
   "id": 1,
   "name": "Scarce Search Results",
   "description": "el formulario de búsqueda usualmente no trae resultados",
   "date": "2018-04-01T14:02:12-03:00", ← fecha en la que fue creado
   "attributes": [{ ← Lista de atributos opcionales para mejorar
        "id": 1,
        "label": "Lista valores posibles", ← Label para mostrar
        "description": "Ingresar separado por comas la lista de posibles valores
        para la búsqueda",
        "attr_type": "T", ← Tipo de atributo [[T - Text, B - Boolean, N - Number,
        D - Date, X - xPath]]
        "date": "2018-04-01T14:03:09-03:00",
       }]
}]
```

Solicitar smell por ID

Devuelve un smell por ID con sus atributos. Estos atributos no son obligatorios, pero si el usuario los completa le permite a Kobold devolver un listado de refactorings más específico para solucionar el problema.

Request

```
GET /smells/api/smells/[smellId]
```

Response

```
{
   "id": 1,
   "name": "Scarce Search Results",
   "description": "el formulario de búsqueda usualmente no trae resultados",
   "date": "2018-04-01T14:02:12-03:00", ← fecha en la que fue creado
   "attributes": [{ ← Lista de atributos opcionales para mejorar
        "id": 1,
        "label": "Lista valores posibles", ← Label para mostrar
        "description": "Ingresar separado por comas la lista de posibles valores
        para la búsqueda",
        "attr_type": "T", ← Tipo de atributo [[T - Text, B - Boolean, N - Number,
        D - Date, X - xPath]]
        "date": "2018-04-01T14:03:09-03:00",
       }]
}
```

Solicitar lista de atributos de smells

Listado de todos los atributos asociados a todos los smells. El listado se presentará por atributo y se mostrará a qué smell pertenece.

Request

```
GET /smells/api/attributesSmells
```

Response

```
[{
  "id": 1,
  "label": "Lista valores posibles",
  "description": "Ingresar separado por comas la lista de posibles valores para la
búsqueda",
  "attr_type": "T",
  "date": "2018-04-01T14:03:09-03:00",
  "smell": {
       "id": 1,
       "name": "Scarce Search Results",
       "description": "el formulario de búsqueda usualmente no trae resultados",
       "date": "2018-04-01T14:02:12-03:00"
        }
}]
```

Anexo B

API Kobold

Solicitar lista de smells

Devuelve el listado de smells disponibles con sus atributos. Estos atributos no son obligatorios, pero si el usuario los completa le permite a Kobold devolver un listado de refactorings más específico para solucionar el problema.

Request

GET /api/smells

Response

```
[{
        "id": 1, ← ID del smell en la base de datos de Kobold
        "name": "Distant Content",
        "description": "Users often follow a path of many navigation steps too quickly to get to a point. They do not seem to stop in the middle nodes and see their contents.",
        "attributes": []
      },{
       "id": 2, ← ID del smell en la base de datos de Kobold
       "name": "Overlook Content",
       "description": "Users often follow a path of many navigation steps too quickly to get to a point. They do not seem to stop in the middle nodes and see their contents.",
        "attributes": [{ ← Lista de atributos opcionales para mejorar
             "id": 1, ← ID del atributo en Kobold
             "label": "upwards", ← Label para mostrar
             "description": "True when quick scrollings happened from bottom to top.",
             "attr_type": "B", ← Tipo de atributo [[T - Text, B - Boolean, N - Number, D - Date, X - xPath]]
     }]
    }
]
```

Solicitar smell por ID

Devuelve un smell por ID con sus atributos. Estos atributos no son obligatorios, pero si el usuario los completa le permite a Kobold devolver un listado de refactorings más específico para solucionar el problema. Cabe destacar que el smells puede tener o no atributos asociados.

Request

GET /api/smells/[smellId]

Response

```
{
  "id": 2, ← ID del smell en la base de datos de Kobold
  "name": "Overlook Content",
   "description": "Users scroll to an area too quickly. It is possible that they
don't read the contents between the top of the page and said area."
  "attributes": [{ ← Lista de atributos opcionales para mejorar
       "id": 1, ← ID del atributo en Kobold
       "label": "upwards", ← Label para mostrar
       "description": "True when quick scrollings happened from bottom to top.",
        "attr_type": "B", ← Tipo de atributo [[T - Text, B - Boolean, N - Number,
D - Date, X - xPath]]
     }]
 }
```

Solicitar lista de refactorings para determinado Smell

Devuelve una lista de refactorings posibles para resolver el smell especificado por parámetro. Un refactoring puede solicitar uno o varios atributos, los cuales son obligatorios para poder solicitar el código a ejecutar.
Existe la posibilidad de que el listado de ***additionalAttributes*** contenga una lista compuesta, es decir, un array de array de objetos donde cada array interno corresponde a un conjunto de atributos (1 a N elementos) que pueden ser repetidos para la generación del código del refactoring.

Request

POST /api/smells/[*idSmellKobold*]/refactorings

Body

```
{
  "smellId":1, ← IdKobold del smell
  "attributes":[{
        "attributeId":132, ← IdKobold del atributo
        "value":"/body/div[0]/h1" ← Valor ingresado para dicho atributo
      }]
}
```

Response

```
[{
  "id":4, ← idRefactoringKobold
  "name":"TurnAttributeIntoLink",
   "description":"Integer posuere erat a ante venenatis dapibus posuere
velit aliquet.",
  "additionalAttributes":[{
        "id" = "1"
        "label" = "home"
        "description" = "página principal"
```

```
            "type" = "X"← Tipo de atributo [[T - Text, B - Boolean, N -
        Number, D - Date, X - xPath]]
        },[[{"id" = "2"
            "label" = "Section"
            "description" = "xPath to the section"
            "type" = "X"← Tipo de atributo [[T - Text, B - Boolean, N -
        Number, D - Date, X - xPath]]
            },
            {"id" = "3"
            "label" = "Section name"
            "description" = "Section's name"
            "type" = "T"← Tipo de atributo [[T - Text, B - Boolean, N -
        Number, D - Date, X - xPath]]
          }]]
}]
```

Solicitar refactoring para ejecutar

Devuelve el código javascript para que sea ejecutado del lado del cliente. Se debe enviar el listado, en caso de corresponder, de atributos obligatorios.
Si el listado de atributos contiene atributos compuestos y repetibles estos deberán ser duplicados la cantidad de veces que sea necesario dentro de los corchetes. Los atributos simples deberán ir entre llaves. En el ejemplo del request a continuación se entiende,

- El atributo con id = 1 es un atributo simple.
- El conjunto de atributos con id=2 y id=3 corresponden a atributos dinámicos.

Tener en cuenta que el código devuelto exige jQuery en el cliente.

Request
POST /api/refactorings/[*refactoringId*]
Body

```
{
  "attributes":[{
          "id" = "1" ← ID smell en kobold
          "value" = "/home"
        },[[{
            "id" = "2" ← ID smell en kobold
            "value" = "/home/div/"
          },
          {
            "id" = "3" ← ID smell en kobold
            "value" = "section name"
            },
          {
            "id" = "2"
            "value" = "/home/div/p"
          },
          {
```

```
                "id" = "3"
                "value" = "section name 2"
            }]]
        ]
    }
```

Response

```
    {
        "code":"/********* LinkToTop ***********/
                                    $('body').append('<a          id='scroller'
style='display:block;position:fixed;bottom:30px;right:30px;width:35px;height:35px;
cursor:pointer;background:
url(https://selfrefactoring.s3.amazonaws.com/resources/refactorings/totop.png)
no-repeat;display:none'></a>');$(window).scroll(function () {
                                        if  ($(this).scrollTop()  >  0)  {
$('#scroller').fadeIn();}
                                    else { $('#scroller').fadeOut();}
                            });

                            $('#scroller').click(function () {
                                $('body,html').animate({ scrollTop: 0}, 400);
                                return false;
                            });"
    }
```